THE DEVELOPMENT OF AN INTEGRATIVE STRUCTURE FOR DISCRETE EVENT SIMULATION, OBJECT ORIENTED MODELING AND EMBEDDED DECISION PROCESSES

By

SEREF CEM KARACAL

Bachelor of Science
Middle East Technical University
Ankara, Turkey
1982

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1986

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1991

THE DEVELOPMENT OF AN INTEGRATIVE STRUCTURE FOR DISCRETE EVENT SIMULATION, OBJECT ORIENTED MODELING AND EMBEDDED DECISION PROCESSES

Thesis Approved:

Joe H. Mize
Thesis Advisor

Carl B. Estes

Kenneth E. Case

M. P. Lonell

C. M. Baum

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my gratitude to the faculty and staff of the Industrial Engineering and Management Department for making our stay at Oklahoma State University a rewarding and enjoyable learning experience. I wish to express my sincere gratitude to my major advisor Dr. Joe Mize, not only for his guidance and advise throughout this study, but also for the support, understanding and friendship he has offered me during my difficult times. He has my foremost respect for setting a great example of professionalism and dedication. I would also like to thank Dr. Mize for the opportunity of being part of CIM center where most of the ideas of this study are shaped.

I would like to extend very special thanks to my committee members for their assistance and guidance during my years at OSU. Specifically, I would like to thank Dr. Kenneth Case, Dr. Carl Estes, Dr. Palmer Terrell, and Dr. Charles Bacon, each of which taught me and deserve many thanks.

I would like to offer special thanks to my friends Terry Beaumariage, Pablo Nuno, Chuda Basnet, Phil Farrington, and Dave Pratt for their friendship and inputs to this work.

I wish to express my deep appreciation to my mother Gonul and to my brother Can for their continuous encouragement and support. Without the loving support and understanding of my wife Zeynep and my daughter Sila this accomplishment would not have been achieved.

I would like to dedicate this work to the memory of the greatest man of my world, my father Dr. Orhan Karacal whom I unexpectedly lost during this study. He was our best friend, teacher, and guide. I have the deepest respect and admiration for him and I will live by his teachings.

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
II.	PROBLEM STATEMENT	6
III.	RESEARCH OBJECTIVES	11
IV.	BACKGROUND OF THE STUDY	14
	Introduction	14
	Simulation Modeling and Formalism	15
	Object Oriented Programming	22
	Benefits of OOP Approach to Simulation	24
	Object Oriented Simulation Systems	26
	Artificial Intelligence (AI):	28
	A Brief Review	28
	AI and Simulation	30
	Expert Systems and Simulation	32
	Knowledge Based Simulation	35
	New Simulation Systems	39
	Intelligent Simulation Systems	45
V.	A FORMALISM FOR MODELING MULTIPLE LEVEL SYSTEMS	53
	Introduction	53
	Definitions	55
	System Sophistication Levels	58
	System Entities	82
	Decision Making Entities	84
	Data Driven Physical Entities	88
VI.	INFORMATION AND KNOWLEDGE PROCESSING	93
	Introduction	93
	States and Actions	94
	Information Processing	95
	Knowledge Processing	102
	Knowledge Base	105
	Goal Regression	110

Chapter Page

VII. GOAL DECOMPOSITION FOR MULTIPLE LEVEL DECISION PROCESSING 119

Introduction 119
Simulation Objectives and System Objectives. . 121
Formal Representation of System Objectives . . 123
Formal Language Theory 124
Goal Decomposition 126

VIII. IMPLEMENTATION IN OBJECT ORIENTED PROGRAMMING 135

Introduction 135
Smalltalk-80 Framework for Simulation 136
Formalism and Smalltalk-80 Implementation: . . 142
Formal Structures and Software Entities . 142
Smalltalk-80 Classes Defined 156
Modifications Made to Smalltalk-80 and Simulation 181
Intelligent Manufacturing System Entities and Knowledge Bases 185

IX. EVALUATION OF THE APPROACH THROUGH AN EXAMPLE MODEL AND ANALYTIC HIERARCHY PROCESS 191

An Example Manufacturing System 191
Analytic Hierarchy Process 213

X. CONCLUSIONS AND RECOMMENDATIONS 228

Conclusions 228
Contributions 232
Recommendations 234

BIBLIOGRAPHY 236

APPENDICES (*)

APPENDIX A - SMALLTALK-80 CLASSES DEFINED

APPENDIX B - MODIFIED SMALLTALK-80 CLASSES

APPENDIX C - EXAMPLE HUMBLE KNOWLEDGE BASES

APPENDIX D - RESULTS OF EXAMPLE SIMULATION RUNS

APPENDIX E - AHP CALCULATIONS

(*) Available in the School of Industrial Engineering and Management Library at Oklahoma State University.

LIST OF TABLES

Table		Page
I.	Summary of Simulation Runs	206
II.	Node 1.1 - Best Simulation Paradigm	220
III.	Node 2.1 - Model Effectiveness.	220
IV.	Node 2.2 - Model Developer's Potency and Modeling Effort.	221
V.	Node 2.3 - Model Execution Performance	221
VI.	Node 2.4 - Model's Degree of Correspondence to real System	222
VII.	Node 3.1 - Formal Modeling Structures / Modeling Methodology	222
VIII.	Node 3.2 - Model Flexibility	223
IX.	Node 3.3 - Output Provisions	223
X.	Node 3.4 - Execution Speed	224
XI.	Node 3.5 - Physical, Information, and Control Components	224
XII.	Node 3.6 - Primitive Modeling Constructs	225
XIII.	Node 3.7 - Non-Programmed Decision Facilities.	225
XIV.	Final Solution Weights	226

LIST OF FIGURES

Figure		Page
1.	A Taxonomy for Combinations of Expert Systems and Simulation	33
2.	Overall Structure of KBS	36
3.	A Suggested Architecture for an OOP Based Simulation Environment	43
4.	Hierarchical Semantic Definition of Knowledge	49
5.	Definition of Data	56
6.	System Sophistication Levels	59
7.	Information System	75
8.	Overall Structure of the Formalism	81
9.	Decision Making Entity Taxonomy	87
10.	Physical System Taxonomy	90
11.	Decision Making Entities of a Manufacturing System	92
12.	State Transition Diagram of a Machine Tool	97
13.	Control Object Operation	98
14.	Exception Events Control Object	101
15.	Knowledge Processing	107
16.	Example Knowledge Processing Scheme	114
17.	Goal Regression Search Space Example	116
18.	SystemStructure Tree	147
19.	OrderDataBase Tree	152
20.	Order Flow and Communication Diagram	155

Figure Page

21. Example System's Physical Configuration 193

22. Example Product Structure 194

23. Example Product Structure 194

24. AHP Hierarchical Diagram 219

CHAPTER I

INTRODUCTION

Simulation is gaining popularity as a decision making tool in today's highly complex systems. As the size and the complexity of the problems coming from those systems increase, simulation is replacing the classical optimization approach. The difficulty of mathematically representing and solving complex problems is forcing decision makers to use simulation as the last resort. In cases where mathematical modeling is made feasible through simplifications and assumptions, analytical methods often result in a collection of suboptimal solutions. On the other hand, while gaining wide spread acceptance, simulation methodology still suffers from its traditional deficiencies. It is clear that the recent trend from analytical tools to simulation is not due to the increased power of simulation but due to the increased consensus on the inappropriateness of analytical tools and increased computer literacy among decision makers.

As is well known, simulation does not provide exact or optimal solutions to problems but helps predict the behavior of the system under various courses of action. Thus, simulation can be perceived as a satisfycing oriented

dynamic approximation method for decision making. Bearing the actual reason for simulation's increasing popularity in mind, there is an intense interest among researchers in improving simulation in several different aspects. Although the research interests show great variety, there are some main stream approaches being followed. These approaches can be grouped into the following two major classes:

I - Extensions of traditional simulation languages

II - Revolutionary approaches to simulation

The following paragraphs briefly explain the above classes by identifying their general characteristics.

I - Extensions of traditional simulation languages:

These software systems mainly consist of front and/or back end modules built on present simulation languages for the purpose of model development, automatic program generation, simulation analysis, and decision support. The main idea lying behind these systems is to take full advantage of some well established and powerful aspects of current simulation languages. In other words, the approach is to use a popular simulation language as the base for simulation execution and statistics collection with some additional layers of software that aid the user in modeling and analysis. At present, most of the development work being done is in the front end of the simulation, targeting model building activities. The recent developments in computer hardware technology have facilitated the use of graphics in simulation which have provided the opportunities for

graphical simulation modeling. The systems in this group can be further divided into two main groups.

1 -Systems based on classical representation and programming techniques: These are mainly the graphics and animation tools that are used to enhance simulation results and model development. Some of them also provide features for debugging, monitoring simulation execution and interactive model modification.

2 -Systems which are the combination of modules and ideas from present simulation languages as well as new programming and representation schemes: These can be classified as expert systems in conjunction with the traditional simulation paradigm.

II - Revolutionary systems: These systems are based on new programming paradigms and knowledge representation methods along with new perspectives in viewing and analyzing systems. Within this class, the major interest areas are: Object Oriented Programming (OOP), Logic Programming and expert systems, Distributed Simulation, and Knowledge Based Simulation (KBS). The natural link between OOP and discrete event simulation is addressed extensively in various papers (Adiga, 1986, Roberts, 1988) and will be summarized in the following chapter. Logic programming and expert systems, which are suitable for programming and processing of mainly qualitative knowledge, comes into use where this type of knowledge is appropriate for representing the structures, relations and behaviors that exist in the system being

considered. The main motivation behind distributed simulation is to reduce the computational time required to simulate large complex systems with a very large number of transactions. The processing of simulation in distributed systems requires the synchronization of the messages passed between simultaneously operating physical entities. A Knowledge Based Simulation (KBS), sometimes referred to as "intelligent simulation system", is an environment which provides facilities for interactive model creation and alteration, simulation monitoring and control, graphical display, and selective statistics collection. An ideal KBS system performs these functions automatically by utilizing various knowledge bases interactively.

In addition to the recent developments in simulation that are briefly outlined above, two other areas, namely, systems simulation formalism and decision making process (although not to be investigated in an analytical sense), will be the topics of interest in this research.

Within the context of discrete event simulation, formalism can be defined as the set of conventions for the construction of discrete event simulation models. In broad terms, it gives a definite form to how and what can be expressed about a system to be modelled. With the use of formalism one can specify a discrete simulation model in a precise and unambiguous manner. Formalism uses set theoretic and system theoretic concepts for the abstraction of real systems that, in the end, generate a uniform

convention of communication.

Decision making is a characteristic of purposeful systems. Decision making processes usually involve several identifiable stages. The first stage is the recognition that a decision problem, an obstacle to achieving a goal, exists. The remaining stages are: identification of alternatives, evaluation of these alternatives, selection of the one "best" alternative, and implementing the decision. Formal analysis, while playing a role throughout the process, is used mainly in the evaluation and choice stages. Traditionally, some additional 'in between' stages are implicitly involved in the decision making process, e.g. identifying the criteria and experimenting.

Furthermore, nonmonotonic reasoning systems that are defined below hold some potential benefits in expressing the logic involved in most real life decision making situations. A logic system in which a conclusion stands no matter what new axioms are added is called monotonic. Classical formal logic systems are all monotonic. In cases where new information causes an old conclusion to be withdrawn, the entire reasoning system is classified as nonmonotonic.

The areas that are briefly introduced above define the general aspects involved in this research. Due to the fragmented nature of research efforts and communication gaps between the areas, there is a strong need for the development of a unified framework for relating these various aspects of simulation modeling.

CHAPTER II

PROBLEM STATEMENT

During the traditional model development process various system assumptions are hardwired into the model code. The modification of these assumptions usually requires major model overhauls with extensive programming efforts, resulting in minimal ease of model development and code reusability. In addition to that, present modeling tools force the model developer to think in terms of an abstract set of reusable blocks which represent various operations that are typically found in simulation modeling. Besides the difficulty of modeling, traditional simulation languages lack representation of complex behavior such as decision making. Representation of decision logic within the simulation model may require symbolic processing which is only available in a few of the commercially available simulation systems. Since the detailed operational logic is also hardwired into the model in present systems, experimenting with different decision strategies results in major model modifications.

The main purpose of traditional simulation is to make inferences about the behavior of certain dependent random

variables. At present, the analysis of the results is an off-line, time consuming activity that is done through the use of statistical analysis techniques. Furthermore, present simulation methodology focuses on very few aspects or parameters of the system if not on a single aspect. In other words, the ultimate result of a traditional simulation is a proposed solution that is implemented by setting the values of certain controllable variables.

The question of how to incorporate artificial intelligence concepts into simulation is one of the current research interests. There have been several recent attempts to develop intelligent discrete event simulation models. Unfortunately, most of the recent intelligent simulation systems are also developed in an add-hoc manner without actually formalizing the so called "intelligent simulation systems" concept. Some researchers have perceived intelligent simulation systems within the context of expert systems and have translated some of the simulation expertise into the software. Others have focused totally on qualitative aspects of the systems overlooking the power of quantitative information in analyzing and controlling systems. The reason for the later was partially due to the desire of taking full advantage of symbolic processing languages that are used as the base language for model development and simulation processing.

As stated earlier, simulation suffers from the fact that models developed through the use of current modeling

tools and methodology usually do not accurately represent reality no matter how detailed they are. The reason for this is the inadequacies of present approaches in expressing the rules and heuristics employed by decision makers of the system in the conduct of their activities. Therefore, the capability to represent and process the decision maker's activities adds an extra yet very important dimension of reality in dynamic models of real systems. Traditional modeling tools and methodology are poorly qualified for modeling cognitive processes. In reality, it is the cognitive processes that strongly affect the dynamic behavior of the physical system. Models with cognitive capabilities are essential in expressing the dynamics of systems involving one or more intelligent and rational decision making entities. Detailed modeling of systems whose real behavior is influenced by many internally originating decisions can only be achieved by explicitly representing the decision making processes along with their information components involved.

Traditional modeling approaches allow very little internal decision making. Such situations are usually modeled by predefined structures. For example, DETECT, SELECT, and MATCH nodes along with hardwired branching decision rules can be used in SLAM to handle some of the internal decisions. In reality, each decision making situation encountered may involve several rules, procedures, and heuristics that must be intelligently searched and

applied as required. Also, the system of interest may contain several decision making situations like this that need to be modeled. Examples of such systems are very common in reality, e.g. a multi-level system in which several levels of management are responsible for control of a manufacturing process.

Within the context of the above discussion, the present modelling approaches must be revised to include cognitive processing capabilities. A different approach that unifies new programming paradigms like Object Oriented Programming and Logic Programming and new methods for knowledge representation could possibly provide an environment that opens new possibilities for building powerful and realistic simulation models. In this new perspective, the modeler will perceive the system of interest in terms of several physical entities, augmented by some number of decision making elements, each capable of making rational decisions that affect the behavior and contribute to the collective state of the system. This view will also allow an efficient marriage of artificial intelligence and simulation by integrating the qualitative and quantitative knowledge to realistically model and simulate the system. In contrast to the traditional view, the result of simulation in this perspective will be a plan in terms of time phased decisions that need to be made to drive the system towards a desired state. Graphical user interfaces for modeling, monitoring,

and output analysis will be essential parts of such a simulation environment. The modularity of the user's conceptual model in terms of physical and decision-making entities will be extremely important in addition to the modularity of computer executable code. Furthermore, the efficiency and effectiveness of the simulation will depend on the system's capability of modeling and abstraction of both physical and decision processes at different hierarchical levels.

In summary, the specific problem to be addressed in this research is:

Current modeling approaches do not provide for an adequate representation of information processing and decision making elements.

CHAPTER III

RESEARCH OBJECTIVES

The main purpose of this research will be the development and evaluation of appropriate representation schemes and procedures to explicitly express information and decision elements and their dynamic execution within simulation models. Specific research objectives are defined as follow:

Objective 1:

Develop a formalism for intelligent discrete event simulation. This will be the integrated representation of qualitative and quantitative knowledge together with formal representations of data, information, relations, and control within simulation models. The schemes developed here will be the building blocks for simulating decision making processes.

Objective 2:

Develop a prototype knowledge processing scheme in conjunction with prototype heuristics, rules, and procedures that is capable of carrying out dynamic control during the

execution of the simulation. This knowledge processing mechanism may have certain nonmonotonic aspects, allowing previous conclusions to be deleted as required due to the availability of a new piece of information.

Objective 3:

Translate and partition the simulated system's objectives into a form that is understandable and manageable by lower level control structures. This will be the definition and distribution of tasks among decision making entities to achieve a common objective. Each decision making entity will conduct its own cognitive process realizing the multi-criteria nature of this process.

Objective 4:

Implement the outcomes of Objectives 1, 2, and 3 in a manner that leads to the development of a prototype methodology for conducting simulation. This methodology will allow the inclusion of non-deterministic endogenous decisions in simulation models. These decisions will be expressed by modeling the authority and responsibility of each decision making entity along with its cognitive process.

Objective 5:

Evaluate the prototype methodology that will be developed in terms of its value and quality in modeling

systems. This will require the definition of tangible and intangible benefits of this new modeling methodology over traditional approaches. This objective will be achieved through the development of measures that allow the comparison of appropriate facets of the proposed and traditional modeling methodologies.

In order to achieve the above objectives the research is divided into five separate phases, each of which corresponds with objectives one through five.

CHAPTER IV

BACKGROUND OF THE STUDY

Introduction

This chapter contains a review of the research being performed in the areas of Artificial Intelligence (AI) and OOP as applied to discrete event simulation methodology. From the extensions of current simulation languages, only the approaches that include either OOP or AI concepts are covered in this literature survey. Thus, the principal emphases in this survey are the new simulation paradigms and simulation environments. First, to keep a better perspective, the systems simulation modeling with its scope, importance and formalism is briefly presented. After a short introduction to the OOP paradigm, its potential use and value in simulation modeling is discussed. Next, the role of AI in simulation, its benefits and limitations are reviewed. Then, various approaches taken to merge OOP and AI into simulation are described. Finally, some specific approaches that are conceived to be the most compatible with the author's ideas are explained.

Simulation Modeling and Formalism

Simulation models are descriptive models. They offer only symbolic representation of some problem space, without giving any guidance on how to search it. Use of descriptive models is therefore an inductive experimental technique for exploring possible worlds through a computational process (Kreutzer 1986). The experimental frame concept that is used by both Zeigler (1984) and Kreutzer (1986) defines a set of circumstances under which a system or model will be observed. Experimental frames define the significant variables and their initial values, inputs, initial state of the system, data collection specifications, and termination conditions. Thus, using simulation involves searching for solutions through a finite number of experiments (runs) to which a model can be subjected.

Simulation models can be used for two different purposes. In the traditional view, modeling is an activity undertaken to increase decision making capability. Models provide the ability to observe the effects of a decision maker's choices before they are actually implemented. In the light of the decision objectives, the most promising alternative is then selected. In this approach, simulation models still include strong analytical characteristics. In such cases, the decision problem is relatively well defined and simulation is selected because either no appropriate

analytical tool exists or none of the available analytical tools is applicable to the decision problem on hand. The power of this type of modeling lies in the accuracy of its predictions about system behavior.

Recently, another type of simulation use is gaining popularity among researchers. In this view, as stated in Baskaran et. al (1984), the purpose of modeling is insight, not numbers. This type of modeling aims at improved understanding of complex systems. The building of a model defines its purpose and provides improved intuition about the system's behavior along with its essential facets and sensitivities. This approach promotes a more explanatory and speculative style of modeling for situations where not much is known about a system's behavior, or problems are relatively ill-defined.

Fishwick (1989a) defines the term "qualitative simulation" as "simulation employing abstract modeling methods" and he investigated the terminology and issues in qualitative simulation. After stressing the importance of the communication problem between the AI and simulation fields, he defines the following methods to reflect the qualitative approach to simulation:

- Relaxation of variable(s) in the system.
- A lumped model derived through either structure preserving simplifications or another valid method of process abstraction.
- A visual approach to model design and analysis.

- A method for inferring model structure from raw data and sketchy models.
- Liberal use of constraints in pruning the solution space if the simulation model behavior is ambiguous.

Fishwick defines two type of goals for qualitative simulation, which are: (1) simulating human reasoning about a process and (2) developing qualitative models for simulation. After pointing out the artificial distinction between symbolic knowledge (frames, production rules, etc.) and mathematical knowledge (difference equations, etc.), he explains the cases where a symbolic simulation system is still needed. These instances are:

- To represent human thought about a process
- Symbolic system is computationally attractive
- Symbolic system is used as an instructional device

Modeling and simulation in general still lack sound theoretical and methodological foundations. In current simulation methodology, despite the existence of well developed tools and their generalized building blocks, modeling is still carried out in an ad hoc and intuitive manner. The reason for this fact is that there is not a well accepted formalism and a methodology defined for modeling. The building blocks and primitives of traditional simulation tools themselves are usually developed by an experience based approach. The efficiency and accuracy of modeling depends on how easily the modeler can associate his

ideas or vision of the system under study to the tool provided. As a result, the success of the modeling methodology depends on how sound the theoretical base is. The simulation modeling formalism stresses the importance of methodology and notation by using concepts from set theory and systems theory.

Systems theory is a scientific discipline whose primary concern is to provide problem solving methods and tools. The first attempts to unify system theory and simulation modeling originates from General Systems Theory Implementor Language (GEST) developed by Oren (1984b). Later, an advanced version of GEST with its high level development shell, Modeling Advisor for GEST (MAGEST) was designed by Oren and Aytac (1985). MAGEST provides a means for specifying hierarchical models. Another development in this direction was the General Systems Problem Solver (GPSP) for inductive modeling which was developed by Klir (1984). Pichler (1984) developed the Computer Aided Systems Theory (CAST) by providing method banks for computer aided problem solving. Rozenblit (1988) gives a summary of the work that has been done in this area.

The usefulness of system theoretic concepts, when modeling and simulating complex systems, became apparent by the work of Zeigler (1976, 1984, 1987). In this view, which will be explained in detail, models are conceived as means of specifying systems and a two dimensional approach is defined for such system specification. The hierarchical

levels of system specification constitutes one dimension while the formalism for system specification forms the other dimension. Starting with the notion that a system is a collection of interacting component systems, Zeigler provides a hierarchy of system specifications with the morphism concept that enable comparisons between systems specified at any level of abstraction. The levels are defined as follow:

1 - Input/Output Relation Observation (IORO). This is a classical example of a black box.

2 - Input/Output Function Observation (IOFO). For each input function of a given IORO there exists exactly one output function.

3 - I/O System (S). In addition to input and output sets, a set of states and a state transformation mechanism have to be defined at this level.

4 - Structured System. System specification at this level is the same as that at level 3 except each of the sets and functions are structured. In other words, they are made more concrete by being represented as cross-products of more elementary sets and functions.

5 - System Specification NET (coupled system). NET denotes a network of system specifications consisting of a family of systems and a coupling mechanism. This specification is the basis for a hierarchical form of model construction.

Although Zeigler (1984) developed Discrete Event Systems

Specification (DEVS) formalism for the formal representation of discrete event systems, the above decomposition of system specifications is independent of any particular modeling formalism. In other words, any formalism can be employed to specify a system at any level.

DEVS is a structure:

$$M = < X, S, Y, \delta, \sigma, ta >$$

where

X is the external event set (input set)

S is the sequential state set

Y is the output set

δ is the transition function

σ is the output function

ta is the time advance function

DEVS specifies an I/O system:

$$S = < T, X, \Omega, Q, Y, \delta, \sigma >$$

where

T = real numbers for time

$X = X_{DEVS} \cup \{0\}$ (0 represents no event)

Ω = set of discrete event segments over X

The state set is defined as:

$Q = \{ (s,e) \mid s \in S, 0 \leq e \leq ta(s) \}$ where, ta: $S \rightarrow R^{+}_{0,\infty}$ and (s, e) is a total state pair, where s is a sequential state and e is elapsed time in state s. The transition function consists of two pairs:

δ_{int}: S --> S internal transition function

δ_{ext}: Q x X --> S external transition function

DEVS is closed under coupling, enabling the construction of hierarchical DEVS network specifications.

Using the DEVS formalism and multifaceted modeling methodology (Zeigler, 1984) which recognizes the existence of multiple objectives and models in a simulation study, Zeigler implemented a PC-Scheme based software employing the OOP approach (Zeigler, 1987). This environment, which will be discussed in following sections, provides hierarchical, modular specification of discrete event models allowing the development of a reusable model base.

The Product Automaton (PA) formalism (Portier 1987) is another formalism developed for discrete event systems. It is based on a decomposition tree that describes the decomposition of a system into its subordinate systems. PA is a structure:

$$PA = < N, E, r, \{M_i\}, \{Z_{i,j}\} >$$

where, $< N, E, r >$ is a rooted tree which consists of N (node set), E (edge set, $E \subset N \times N$) and r (root, $r \in N$). $M_i = < X_i, S_i, \delta_i, ta_i >$, $(i \in N)$ is a component which is made up of

X_i = set of inputs to i

S_i = state set of i

δ_i = state transition function of i

ta_i = natural update time of i, $ta_i \in R^+_{0,\infty}$

$Z_{i,j}$ is the set of functions defined over edges which translates the input for a component to its subcomponents and is defined as:

$$Z_{i,j}: X_i \times S_i \rightarrow X_j \text{ for all } (i,j) \in E$$

State transition in PA formalism can occur in two forms:

1 - An atomic component has state transitions solely based on input received and on its internal state.

2 - A product (a component that has factors) changes state based on the state of its factors (components that are a part of some decomposition).

Another formalism, Discrete Event Simulation (DES) (Radiya and Sargent, 1987), is based on the following six elements: a model specification language, its semantics, model (its structure and behavior), specification of the validity of a model, proof system for model correctness, and underlying modeling methodology. This formalism proposes ROBS (Rules and Objects Based Simulation) (Radiya and Sargent, 1987) language as the model specification language and introduces the concept of 'agent', which is a function from one model state set to another model state set, and the concept of 'synchronization expression', which is an expression containing predicates defined on the model state with a time indexing element.

Object Oriented Programming

In object oriented programming (OOP), a software system is composed of interacting "objects". Objects are the combination of the attributes of both data and procedures of traditional computer programming. Unlike the conventional

computer programming paradigm which is based on active procedures acting on the passive data that is passed to them, OOP employs a data or object centered approach. In OOP, data is active in the form of an object and performs operations on itself. Each object represents some abstract or physical entity in the problem being solved and is a complete description of the entity, including the data structures to define its structure and state and methods or procedures to define its behavior.

An object is an instance of a "class". The class object provides all the information necessary to construct and use objects of a particular kind. In other words, objects that have things in common are abstracted into a class. Each instance object is a member of one class and a class may have multiple instances. In addition to the data and the way it is stored, a class also provides storage for the methods which are simply procedures that are invoked by sending messages.

In order to call a computer programming language as an object oriented language that language must exhibit four characteristics. These four features: Encapsulation, Message Passing, Inheritance, and Dynamic Binding form the foundation of Smalltalk, which is the original OOP language.

Encapsulation means that data is encapsulated inside an inviolable shell along with the methods required to use it. The only legal way to access the data is through the use of these methods.

Message passing is a natural result of encapsulation. It is similar to a function call in procedural programming. Messages are the carriers of all the interactions between objects.

Inheritance enables creation of classes in a hierarchical tree structure, allowing new objects to be specializations of other objects. A subclass obtains its essential features from its parent class via inheritance but it can also acquire its own characteristics. The value of inheritance in programming is that a new class needs only be specified by how it differs from an existing class rather than being completely redefined.

Dynamic binding, sometimes referred to as late binding, is the process of associating the data and the procedure during run time. It means that references are symbolic and a method can be compiled without compiling all of its callers. The same symbolic names can be used regardless of the type of the object and a single message can invoke several methods. In the OOP environment this characteristic is known as polymorphic behavior and it allows code that is written to be independent of the receiver.

Benefits of OOP Approach to Simulation

The major benefit of object oriented systems is the design philosophy they bring to simulation. OOP provides encapsulation that helps to modularize a problem in its

early stages of analysis. It requires the user to identify the principle components of a system and to specify their structure, behavior and interactions. The object oriented view yields a natural decomposition of a system. Additionally, the object oriented approach allows simulations to become extensible. Existing models can form the basis for new ones and existing concepts can be enhanced to handle new systems. Inheritance permits new objects to be defined from existing ones by just describing the difference. Old models become reusable because their methods and objects continue to be useful. Cox (1987) also points out that the OOP approach causes a substantial reduction in the amount of the resulting code. This implies that a single person can manage more complexity.

Since objects in most simulations tend to be physical and real, the user can often directly translate his simulation model into a graphical and animated simulation without additional conceptual changes. Thomasma and Ulgen (1987) describe such an approach for simulation of manufacturing systems implemented in Smalltalk.

Because the objects contain their own functionality, intelligence can be built directly into this functionality using the techniques of Artificial Intelligence. Furthermore, this "intelligence" can be updated through simulated experience. Incorporation of AI concepts into the object oriented approach can also provide for an explanation system for underlying condition-action rules (Helman and

Bahugna, 1986).

Finally, the OOP approach provides for a natural basis for concurrent, distributed simulation. In such a system, each object can be assigned to its own processor and work independently until it is needed for some form of coordination (Bezivin, 1987).

Object Oriented Simulation Systems

Many concepts of the OOP paradigm have their origins in SIMULA (Dahl and Nygaar, 1966). Although SIMULA never achieved a large popularity (especially in United States), many of the concepts (instance, class, etc.) introduced in SIMULA formed the foundation of OOP languages such as Smalltalk. So, it is not surprising that OOP languages are good platforms for discrete event simulation. Since some of the Lisp and Prolog based object oriented simulation systems are reviewed in the following sections, only the pure object oriented modeling and simulation implementations will be presented in this section.

SimTalk, developed by Knapp (1987), is a discrete event simulation environment implemented in Smalltalk. SimTalk adds queueing support, statistics collection, simulation graphics and interactive user interface to the features that already exist in Smalltalk (multiple process support, interactive programming, graphics, etc.). A class called SimTalk provides central communication and maintains the

time queue and simulated clock. Another class, SimTalkObject, is used to present the classes of objects to be simulated. There are a large number of other classes in SimTalk that include random number generators, probability distributions, statistics collectors, statistics analysis, etc.

Ulgen and Thomasma (1987) implemented an object oriented simulation system in Smalltalk. In this system, class Simulator handles the initialization of time and event scheduling. A class called Event associates a time with something to be done. Since the system is manufacturing system simulation oriented, the other classes in the system are designed to represent manufacturing system entities such as work parts, work stations, storage facilities, etc.

A SIMULA based simulation system (Nyen, 1987) was developed in the Norwegian Institute of Technology. This system defines three major object groups for the simulation of manufacturing systems: Resource Objects, Entity Objects, and Stationary Objects. In addition to the simulation kernel which actually executes the simulation, five other segments are defined that interface the user to the simulation system. The intelligent front and back end modules that carry out the actual user interface are graphical and interactive.

Among several, some other object oriented simulation systems are: a distributed simulation system (Bezivin, 1987), a C++ based object library for parallel simulation

(Abrams, 1988), an interactive simulator for VLSI design implemented in Smalltalk (Van der Meulen, 1989). Beaumariage (1989) gives a good summary of some other object oriented simulation implementations.

Artificial Intelligence: A Brief Review

Due to the fact that AI is a relatively new area, it is appropriate to present a brief introduction to the main aspects of AI that are relevant to simulation. There are many definitions of the term Artificial Intelligence. Two of the most popular definitions listed in Adelsberg (1986) are given below:

> AI is the study of how to use knowledge to solve problems using computers.
>
> AI is the study of mental faculties through the use of computational models.

Two central goals of AI are to make computers more useful and to understand the principles that make intelligence possible. Within the context of AI, the characteristics of intelligence can be summarized as: use of symbols and abstractions, learning from the environment, use of vast amounts of knowledge, difficulty of characterizing accurately, and constant change.

The strength of AI systems lies in their ability to address problems that are primarily symbol manipulation. They also facilitate dealing with problems that are naturally thought of in terms of rules and heuristics rather

than detailed algorithms. The major weaknesses of AI systems are due to the fact that many aspects of intelligence are still very poorly understood. AI systems also require large amounts of knowledge and computing power.

Lisp and Prolog are the major AI languages. Lisp is a language to support symbolic manipulation. It has a uniform representation which means that programs and data look the same. Prolog has declarative and procedural aspects. Prolog is a generalization of relational data base concepts.

Predicate calculus, semantic networks and frames are among the many knowledge representation schemes used by AI environments. First order predicate calculus with its predicate symbols, logical connectives and quantifiers provides a uniform format for representation. Semantic networks consist of nodes and links. Nodes stand for concepts and links indicate how they are related. The most important relation is the IS-A relation indicating that one concept is a specialized version of another. This allows inheritance of information. A frame is a collection of semantic net nodes and slots (to describe aspects) that together describe the information about a single concept (object).

In AI, a search problem is characterized by an initial-state and a goal-state description. A "move" transforms one state into another, hopefully closer to the goal-state. Among many, depth first, breath first and branch and bound are the most popular search methods. Problem solving

methods are applied when a task can be well defined and a solution is needed. Since many of the search methods are well understood both conceptually and analytically, the real task depends on finding adequate heuristics.

Expert systems are the most commercially successful applications of AI concepts. They consist of programs that use sophisticated problem solving techniques and large knowledge bases to solve problems. They can handle a wide variety of well defined tasks within a specific problem domain.

Artificial Intelligence and Simulation

Recently there has been an increasing interest in the possibilities of incorporating the techniques developed by AI researchers into the modeling and simulation process. Some researchers have taken the approach of developing intelligent, automatic programming interfaces to existing simulation systems. In these extension systems an interactive user interface allows the user to describe the system to be simulated in terms of icons, menus and interrogations. Most of these systems are limited to a specific problem domain such as manufacturing systems (Mellichamp and Wahab, 1987), or AGV's (Brazier and Shannon, 1987). In these examples, the system automatically develops the model and the experimentation takes place in an existing simulation language such as SIMAN, SIMSCRIPT, or SIMULA.

There have been attempts to develop automatic programming systems in which the user enters a natural language description of the system to be modeled in a restricted domain. The system analyzes this input and interrogates for additional input for clarification or completeness, if needed. Then, the system automatically generates the code in an existing simulation language. Although the user interface is usually based on a symbolic processing language, these systems are severely restricted due to the current state of the art in natural language processing.

Besides the above front end systems, there have been several attempts to develop intelligent back ends attached to existing simulation systems that will help the user in analyzing results and making suggestions (Nyen, 1987, Seliger et al., 1981). Most of these systems are goal driven; the model is executed and if the desired results are not achieved, the system suggests modifications. These systems fall into the category of expert systems which are made up of a set of rules.

The major advantages of these extensions of current simulation systems are their ease of development and execution speed. Since these systems still follow the traditional simulation paradigm, the user must still decide upon the scenarios to be run and interpret the results.

Expert Systems and Simulation

One approach to combining AI concepts and simulation is the development of expert systems or rule based systems for simulation. As stated by Shannon (1987), rules can be used for multiple purposes in a simulation environment such as:

- To define the behavior of the model
- To test the model for completeness and validity
- To drive the model towards a specified goal achievement

O'Keefe (1986) gives a good discussion of expert systems and their role in simulation. After discussing the similarities and possible cross-fertilization areas between expert systems and simulation, O'Keefe gives a taxonomy for combinations of simulation and expert systems. One possible way of combining simulation and expert systems is embedding an expert system within a simulation model as in Figure 1a, or vice verse as in Figure 1b. Alternatively, a simulation model could interrogate an expert system (Figure 1c). This style of implementation can be useful where an expert system already exists for part of the decision making within the system simulated. Expert systems that execute and use the results from simulations (Figure 1d) are of increasing interest to knowledge engineering. Rather than testing an expert system on a user or a real environment, the expert system can be tested on a simulation. In many instances both an expert system and a simulation can be used together to do some task (Figure 1e). This type of implementation allows the user to directly interact with both systems.

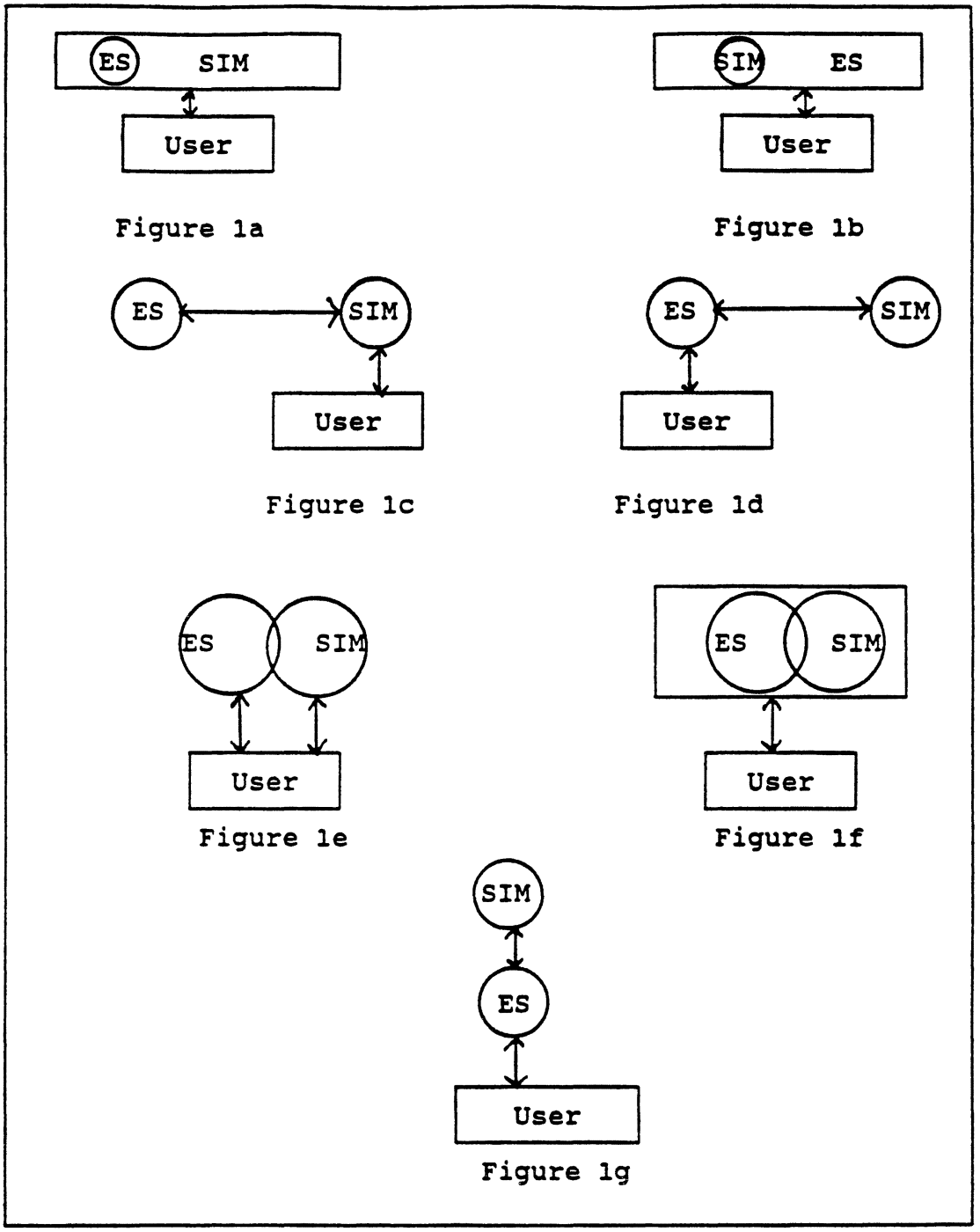


Figure 1, Taxonomy for Combinations of Expert Systems and Simulation (O'Keefe, 1986)

Alternatively, the cooperative system may be surrounded by a large piece of software (Figure 1d) and each may be a part of a large decision support system used directly by decision makers. Knowledge based simulation systems which will be discussed in the following sections fall into this category. Finally, one of the most popular application areas for expert systems is intelligent front/back ends (Figure 1g). This is an expert system that sits between a simulation package and the user, generates necessary code to use the package through a dialogue with the user, and interprets and explains results from the package.

O'Keefe also defines four classes of knowledge when conducting experiments with discrete event simulation models to determine an appropriate design that satisfies one or more objectives. These are:

- Knowledge about the domain in which the model is built, i.e. manufacturing, health care.
- Knowledge about statistics, i.e. how to interpret results, what measurements are appropriate.
- Knowledge about how the simulation (and hence the real system) behaves.
- Knowledge of the language or the environment used to implement the simulation.

Knowledge Based Simulation

Knowledge Based Simulation (KBS) (Reddy, 1987) is a way of combining AI techniques and traditional simulation methodologies. A knowledge based simulation system can be best described with the characteristics it should exhibit. An ideal KBS should

- accept a description of the problem and synthesize a simulation model by consulting an appropriate knowledge base.
- accept a goal in the form of a set of expectations or constraints, select a model at an appropriate level of abstraction, determine the performance metrics, generate a search space of plausible scenarios, execute the simulation model by controlled selection of scenarios, and finally, recommend a scenario that satisfies the stated goal.
- explain the rationale behind why only certain scenarios have been explored and why it recommends a particular scenario.
- learn from experience and disclose its behavior.
- display the resultant model built by KBS with a high degree of perspicuity to increase user confidence.

OOP and Logic Programming (LP) paradigms in conjunction with various inference mechanisms are useful in building KBS systems. Figure 2 graphically summarizes the overall framework of a KBS system. Using the concepts from KBS tools such as LASER/GRAPH provides a convenient mechanism to construct object oriented simulation models.

In a KBS system (Reddy et al., 1986), the schema is the basic unit that represents objects, processes and ideas. A KBS model is a collection of schema representation language schemata (implemented in Lisp) that represent physical and

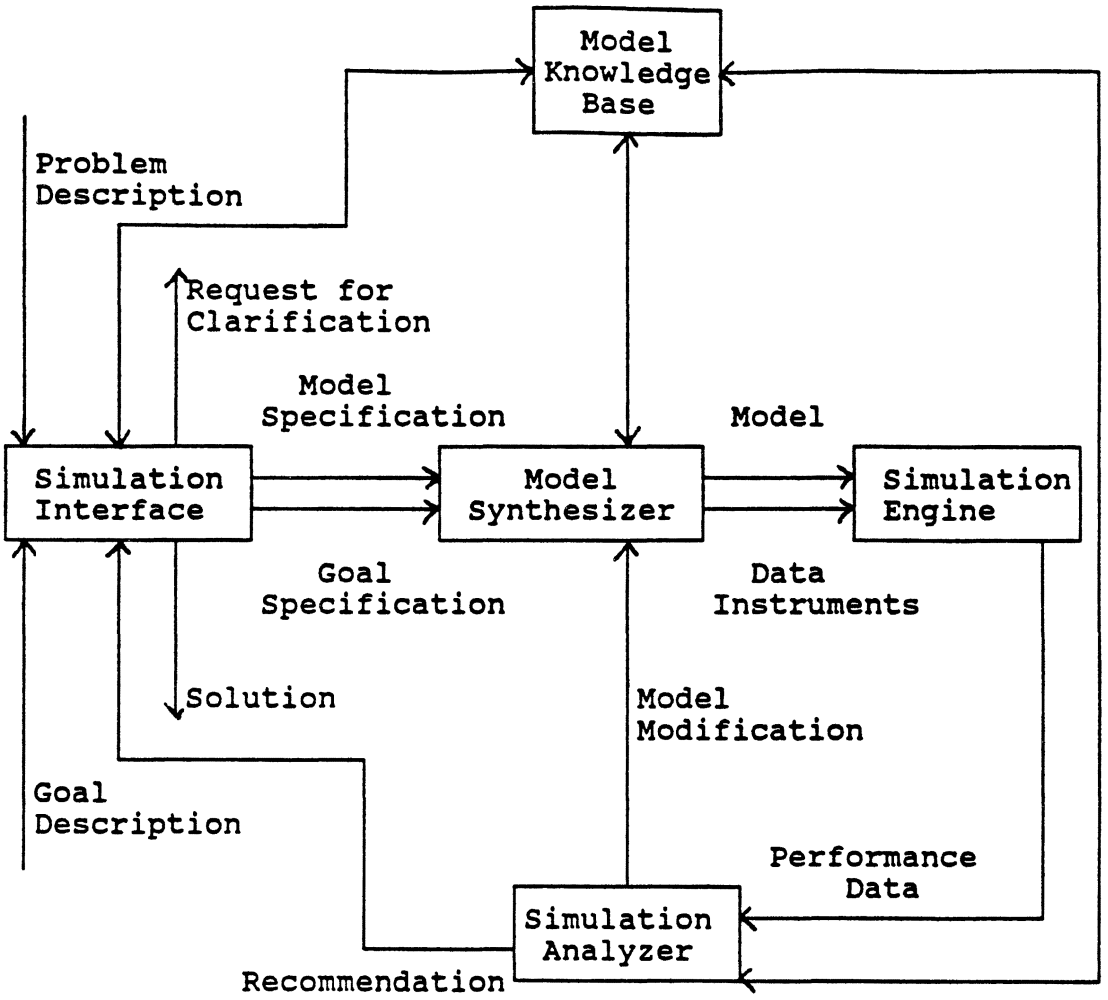


Figure 2, The Overall Structure of KBS (Reddy, 1987)

abstract system entities. Model validation in KBS is also handled by logic programming techniques which are used to specify model consistency and completeness rules. Model simplification techniques for complex simulation models fall into two main categories, static and dynamic. In static techniques, both model structure and model parameters are altered, but event behaviors remain unchanged. Dynamic

simplification techniques alter a model's dynamic processes, redefining some of the event behaviors.

A KBS system employs an event calendar. Event behavior may be expressed as rules to be executed when the event occurs. A KBS system conducts a series of experiments, each being an improvement on past experiments, without human intervention. The improvements are achieved by allowing reasoning between several model scenarios. The evaluation of scenarios is performed by computing a coefficient of goal satisfaction as a weighted average of constraint satisfaction coefficients. The system allows rule based diagnosis along with casual path analysis (for casual explanation) that is executed before statistical analysis. Additionally, although not sophisticated, the system is capable of providing performance analysis, learning, mathematical analysis, predictive analysis, diagnostic and trade-off analysis. The main motivation behind a KBS system is automating the simulation life cycle. Commercial adaptations of the described system are being marketed by the Carnegie Group (Reddy et. al., 1987) under the name of SIMULATION CRAFT, by IntelliCorp as SIMKIT, and by IntelliSys as LASER/SIM.

Oren (1986) defines a set of basic knowledge bases for an advanced simulation environment.

- Model Base: has two parts, the problem independent part and the problem dependent part. The problem independent model base consists of

. a base for icons for graphic modeling

. a base for model modules for computer aided symbolic model processing, and

. a base for already compiled model modules. The problem dependent model base has two parts, defined as individual user files and public files.

- Experimentation Base: consists of files of experimental conditions such as initial and terminal conditions, specification of input, statistics collection and display conditions.

- Numerical Data Base: is also made up of two parts. The problem independent numerical data base consists of an engineering reference data base and software modules to generate numerical data. The problem dependent numerical data base contains modules of input and output data associated with a specific design.

- On Line Documentation Base: provides documentation facilities on knowledge available in the system, on usability of the system, and on actual use of the system.

Oren also defines a set of knowledge bases for meta-knowledge (knowledge on existing processing knowledge). The purpose of these knowledge bases are: modeling, behavior generation, symbolic model processing, and interactive environment.

New Simulation Systems

Realizing the difficulty of classifying new simulation systems developed to merge AI techniques and simulation, an approach which overviews some of the significant studies that have been done in this field is adopted here. Since most of the recent studies aimed at incorporating AI techniques and concepts into simulation include a mixture of ideas from OOP, logic programming, expert systems and knowledge bases, in one form or another, this section is restricted to a brief description of some of these new systems.

A simulation system based on the knowledge based approach is Rule Oriented Simulation System (ROSS) developed in the RAND Corporation (Klahr et al., 1980). It is a Lisp based interactive system which uses the object oriented style of programming. IF-THEN rules describe the behavior of objects and the system aids the user during model execution by displaying a trace of all messages passed during the simulation. Through selective filtering of this trace data, the user can determine if the model is behaving appropriately.

Wales and Luker (1986) define another environment for discrete event simulation implemented in the Ada language. In this system the user interacts with the environment through a multi-level interface using color to distinguish between the different levels. The environment guides the

user through separate model and experiment definition with facilities being available to combine previously verified submodels. During model definition an activity net like diagram is automatically produced and updated on a separate screen to provide an instant picture of the actions of each entity. A discrete event simulation program is generated on request reducing the need to learn a special language.

There have been several efforts to develop Prolog based new simulation systems, mostly from European researchers. At the Hungarian Computer Research Institute, T-Prolog simulation system was developed (Futo et al., 1987). This system combines the time handling primitives of simulation and the symbolic processing of AI into a Prolog super set. The system allows the user to specify the model in first order predicate statements and executes the model with the non-deterministic problem solving methods of Prolog. The user specifies multiple model parameters and goals for the model. Then, the run time interpreter executes the model to find the first set of parameters that meets the goal. Later, the refined version of this system, which is called TS-Prolog, incorporated facilities similar to those found in conventional simulation languages. Processes are started and stopped through the use of predicates and predicates also provide communication between processes. In this system, the model is automatically modified through the use of the backtracking feature of Prolog until the simulation exhibits some desired behavior. TS-Prolog handles both

discrete and continuous simulation. Incorporation of the OOP paradigm into this system is currently being investigated.

Another Prolog based system, developed in Austria, utilizes a concurrent Prolog interpreter similar to TS-Prolog (Adelsberg and Neuman, 1985). In this process oriented discrete event system the user defines the initial structure and the goals. Then, an interpreter implements the "backtracking co-routine" concept.

At the University of Calgary, another concurrent Prolog based system that includes explicit time delay expressions is developed (Cleary and Dewar, 1985). In this system, limited backtracking capabilities enable the exploration of alternate paths for simulation. SIMLOG is also a Prolog based simulation system developed in the University of Lisbon (Rodrigues, 1988). Object-Prolog (Doman, 1988) is another system which tries to integrate logic programming with the object oriented paradigm.

SIMPOOPS (Vaucher and Lapalme, 1987) is a system based on the experimental language POOPS. POOPS tries to combine the best of both logic and object oriented programming. It provides hierarchies of object classes. These objects can function as parallel processes, synchronized by SEND and WAIT primitives. To retain unification and backtracking, a sequencing set and primitives concerned with simulated time are employed.

Another study for the development of a new simulation system is presently being undertaken by an interdisciplinary team of Texas A&M University researchers (Shannon 1987, Adelsberg et al., 1986). The main objective of this effort is defined as "to humanize the simulation environment and process while integrating the functionality, ease of use, ease of model creation, dynamic run time interaction, and model extensibility". This system uses the rule based techniques of ROSS, the knowledge based approach of KBS, and the goal driving mechanism of TS-Prolog. The desired characteristics for this integrative system are listed below:

- Graphical object creation along with a natural language interface aided by an intelligent assistant.
- Simulation model as well as experiments are treated as objects.
- Interactive user interface.
- Run time model modification and display, automatic experimental designs and statistical display.
- Consistency and completeness checks on model, experiments, and objects.
- Goal directed simulation.
- Selection of various abstraction levels of the simulation model and/or experiment.

Figure 3 graphically represents the proposed architecture for such a system.

Robertson (1986) reports the design of an intelligent, rule based simulation environment implemented in OPS5. This system uses production rules as the basis for describing and coding the processes to be simulated. The system uses a forward chaining inference engine, object oriented structuring techniques, augmented by an explicit model of time and contexts, and a modified version of blackboard architecture.

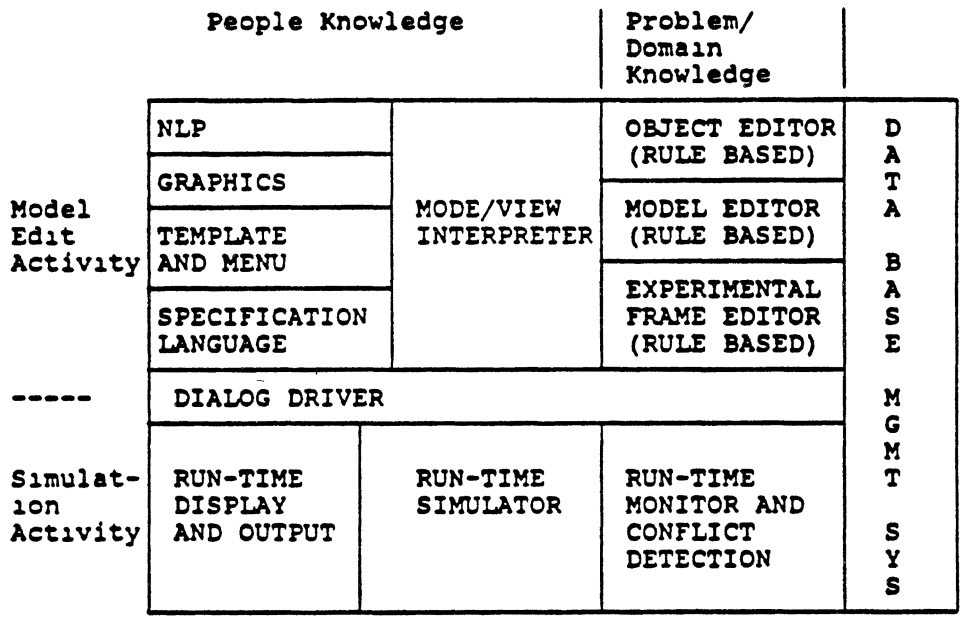


Figure 3, A Suggested Architecture for an OOP Based Simulation Environment (Adelsberg et. al, 1986)

Shaw and Gaines (1986) represent a cognitive science framework for the development of knowledge based systems. Adelsberg (1986) gives a summary of the characteristics of

some of the rule based and object oriented simulation systems. Radiya and Sargent (1987) explore the logic programming paradigm as a computational base for modeling and simulation. In their study, after the suitability of Prolog and LogLisp for simulation are examined, different simulation world views are analyzed to extract the essential features required to implement them in logic programming. LOPPS, a Logic Programming Processor, is the resulting software system which allows modeling and simulation in three different world views.

SIMYON (Ruiz-Mier and Talavage, 1987) is an experimental network simulation environment based on the idea of unifying OOP, logic programming and the discrete event approach for simulation modeling. This system is implemented by defining a library of logic objects in the object oriented, logic programming environment CAYENE (Ruiz-Mier et al., 1985). These objects are analogous to the nodes of network simulation languages and used as building blocks for modeling. The programming environment is based on clauses constructed from Lisp functions. The orientation of this system is to provide the capability to represent complex decisions in a network simulation language like format.

DEVS-Scheme (Zeigler, 1987) is an environment that allows hierarchical, modular discrete event modeling in an object oriented environment. The significance of this system is that its architecture is derived from abstract

simulation concepts associated with DEVS formalism (Zeigler, 1984). It is the only system based on a theoretically well grounded formalism. Closure under coupling property enables the hierarchical construction of simulation models. The experimental frame concept defined in section IV.2 allows the development of simulation test modules in a systematic manner. In the DEVS-Scheme system, a model is viewed as possessing input and output ports through which all interactions through the environment is carried out. This system is implemented in SCOOPS, the object oriented super set of PC-Scheme. The environment also includes some standard class definitions and the execution of simulation is handled by four standard type of messages. The association of System Entity Structure (SES) concept (Rozenblit, 1987), which is a methodology to decompose systems, with DEVS-Scheme is presently being investigated (Rozenblit et. al., 1988).

Intelligent Simulation Systems

The initial incorporation of AI concepts into simulation started with expert systems. This was a natural result and reflection of the fact that expert systems were the most widely accepted implementation of AI concepts. As stated by O'Keefe (1986) later, AI application of expert system concepts into simulation took several different forms, from embedded to front/back ends. Then the term "Knowledge Based Simulation (KBS)" is coined by the Carnegie

Group (Reddy et al., 1986). The concept of KBS was a more comprehensive form of a simulation environment concept which included the classical simulation paradigm instrumented with automatic input, output analysis, and decision support tools. Later as the application of the OOP paradigm to simulation is proven to be appropriate and various OOP languages are made commercially available, most of the previously developed prototype expert and KBS systems can be adapted to this new object oriented approach.

The more recent term being used in this area is "Intelligent Simulation Systems". Unfortunately, like the term AI and our current state of understanding of intelligence, the term "Intelligent Simulation" carries more and different meanings than intended in most of the systems developed. Furthermore, a misconception is created in the science community that every simulation study conducted in Lisp and Prolog is some form of intelligent simulation. Although the term "Cognizant Simulation" is proposed by Oren (1987) to tone down the expectations from intelligent simulation, it is not widely accepted yet.

During this literature review, it is noticed that the term "Intelligent Simulation" is used in two different contexts. Some researchers used the term as a natural extension of KBS systems. In this view , intelligence is associated with the software environment in aiding the user of the software system throughout the simulation life cycle, from modeling to results analysis. The user interacts with

various expert systems and knowledge bases to solve the problem on hand. It is the intelligence of the simulation system that is implied by the term "intelligence" in this approach and the simulation expertise is embedded into the software system. The Carnegie Group's (Reddy et al., 1986), Texas A&M's (Shannon, 1987) and Arizona State's (Hong et al., 1989) systems are well defined examples of this view.

Another and more recent view uses the same term to represent simulation systems that are intended to capture and express the true intelligence which exists in most real world systems being simulated. In this approach, the term "intelligence" corresponds to actual information processing, reasoning and decision making embedded in most physical systems. The term does not represent the intelligence of the software system in facilitating various phases of simulation. Instead, the objective of this second view is to represent real world systems more detailed, accurately and realistically.

Within the context of the second view, the role of AI is mainly to represent and simulate the decision and control structures of real physical systems. Cellier and Zeigler (1987) survey this aspect of AI applications. They identified that two types of knowledge , structural and behavioral knowledge, are the main components of the knowledge about a system. While structural knowledge concerning a system is normally time invariant, and thus static, behavioral knowledge is always time dependent, and

thus dynamic. As stated by Cellier and Zeigler (1987) "rule based models for control are commonly static in nature. The knowledge processed by these systems is mostly structural. If behavioral knowledge is utilized at all, it occurs in the form of statistical data. If time does appear as a variable, it is used only to switch between several (in themselves static) models". They indicated the importance of a methodology for extracting qualitative information from behavioral system data which is usually quantitative.

Hierarchical Segmented Knowledge Bases (HSKB), developed in McDonnel Douglas Astronautics, is an architecture for embedded reasoning in simulation systems (Castillo et al., 1989). This architecture is developed by realizing that when the traditional rule based reasoning paradigm is embedded into simulation, it lacks the organizing principles of object oriented design. HSKB provides a means for building a knowledge base from locally known knowledge segments which are organized according to the taxonomy of objects. This type of architecture results in the object's ability to reason within its specific context and provides an enhanced capability for focusing on local modeling without defining global control strategies. In this system, inference is invoked by special HSKB messages that correspond to specific situations. Thus, only rules pertaining to these situations are inferenced during the simulation. This approach differs from much of the previous work where decision logic is actually hard coded

directly into the object.

The need for explicitly expressing decision making components in simulation modeling is addressed only in a very few articles. Snyder and Mackulak (1988) propose the "decide" node as an efficient means of providing the modeler with the capability to explicitly model non-deterministic decisions. Since accurate representation of real world systems requires qualitative information, these decision nodes are assumed to be based on non-quantitative concepts.

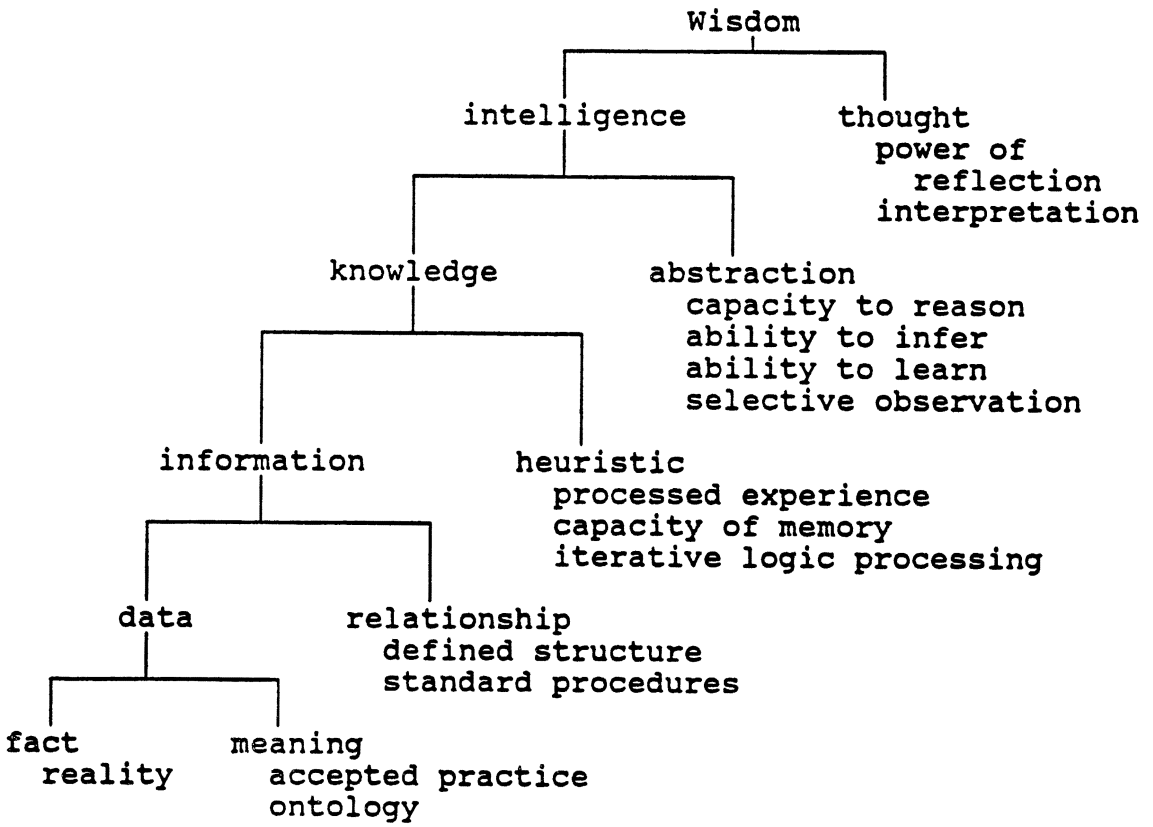


Figure 4, Hierarchical Semantic Definition of Knowledge (Snyder and Mackulak, 1988)

Figure 4 represents the hierarchical semantic definition of knowledge in this view. This study uses a multi-language platform for qualitative evaluation and quantitative analysis in simulation. SIMAN is used for the quantitative part while Lisp and Prolog are used to provide the symbolic processing capabilities. Additionally, a frame based approach similar to OOP is used to represent structures. The investigation of including nonmonotonic reasoning capabilities in the decide node is the future direction of this research.

The research initiated at Los Alamos National Laboratories (Burns and Morgeson, 1988 and Burns et al., 1986) describes the most compatible framework with the author's perception of the "Intelligent Simulation" concept. Therefore, this view will be explained in more detail than the others. The concepts developed in this research are implemented in Intellicorp's Knowledge Engineering Environment (KEE). After noting the inadequacy of conventional simulation languages for representing cognitive processes, it is noticed that the capability to explicitly represent the rules and heuristics employed by managers in the conduct of their activities can enrich the dynamic models of systems. The term "actor" is used to refer to entities which are represented as objects. The actors may be individuals or groups who must make relevant decisions that impact the performance and behavior of the object

system. The term "suit of actors" is used to identify the collection of actor classes. Each actor class is defined in terms of its assets, attributes, vulnerabilities and capabilities. The actors which do not have cognitive capabilities, called pseudo-actors, are categorized into two groups of physical capabilities: transfer and transform.

An actor's cognitive capabilities constitute the decision sets which define the action space for the actor. Two types of activities, physical and cognitive, can be engaged in by an actor. Physical activities are the conventional form of activity around which traditional discrete event simulation has been developed. As stated by the authors, "Cognitive activities are the activities involving some form of intelligent rational decision making. Like physical activities, cognitive activities have finite time durations that may be random, but could also be dependent upon when certain information is available or when a decision becomes urgent". In the conventional simulation paradigm, since neither time duration is associated with decision making processes nor planning for the future is allowed, implicitly represented endogenous decisions are assumed to have zero time durations. This does not allow the simulation of the decision making process and completely ignores the information component that drives the decision making process.

Like activities, events are also of two types, physical and cognitive. Physical activities can be immediately

preceded and followed by cognitive activities and vice verse. Physical events can initiate both physical and cognitive events, while cognitive events can trigger cognitive and physical events. The action space of an actor is dynamic which means that one or more actions from the action space can be removed, one action can be substituted for another, or one or more actions can be inserted into the action space. This characteristic is known as nonmonotonic reasoning in AI literature. The processing mechanism for cognitive activities may be forward-chaining (antecedent reasoning) or backward-chaining (consequent reasoning) when a specific goal has been defined to direct reasoning. When a cognitive event occurs, a decision is made to take certain actions now or in the future. Thus, the result of processing a cognitive event may be the creation of information or a plan.

CHAPTER V

A FORMALISM FOR MODELING MULTIPLE LEVEL SYSTEMS

Introduction

In conventional simulation modeling, data, information, knowledge, and decisions (control) are implicitly represented, if they are represented at all. Furthermore, non-deterministic decisions are usually based on probabilistic and conditional branching rules which remain static throughout the simulation run. Because of the way these non-deterministic controls are represented, models cannot exhibit intelligent behavior such as making state dependent decisions and planning, which are fundamental attributes of purposeful real world systems. For modeling flexibility, modularity, and reality, it is necessary to separate logical events, processes and activities that are taking place in a system from physical events, processes and activities that are also occurring. This observation triggers the search for schemes which allow physical and logical elements of systems to be represented separately. In such a representation, the differentiation and identification of physical and logical objects becomes very

crucial. In this study, it is assumed that all objects that have a real tangible correspondent in the real world system are physical objects. Examples of physical objects in a manufacturing system are machine tools, material handling devices, communication devices, computing devices, etc. Objects that have no real, tangible correspondents in the real world system are logical objects. Some examples of logical objects are data, information, knowledge, plans, procedures, decisions, etc. The set of physical objects can be viewed as system hardware while the set of logical objects constitutes the system's software. Since the formalism presented in the following sections is based on state space representation and differentiates between physical and logical objects, it is fundamentally different from conventional modeling approaches. In the conventional approach, no distinction is made between physical and logical objects and models are based only on physical entities in which logical objects are imbedded implicitly. The following section presents a set of definitions that are fundamental for the formalism proposed.

Definitions

The terms presented below are derived primarily from the studies of Appleton (1984) and Snyder and Mackulak (1989). Since some of these terms are used inconsistently in the related literature, they are clarified for the understanding of the formal representation.

Fact: an observed event, value or symbol without any meaning attached. Facts are the pure symbolic or numeric representations of the above mentioned concepts and they do not convey any information other than abstract representations. Some examples of facts are the numeric string 74075 and the letter M.

Data: an ordered pair of a fact and a meaning. Contrary to common belief, 74075 is not a data. It is a fact, just like M or 85. The problem with facts is that we do not know what they mean. Only when their meaning is provided do they become a datum (Appleton, 1984) (Figure 5). The number 74075 is the zip code of a certain section in Stillwater, Oklahoma when it is given the meaning of zip code; however it might also be the first five digits of a driver license number. M may stand for Male and it is also a letter in the alphabet. The symbol 85 can be an age or can be a test score. Thus, to have a datum two things are necessary, a provided or observed fact and an associated meaning. For each meaning, we can have zero, one, or many facts. The meanings along with facts , not facts alone, define one's

concept of reality.

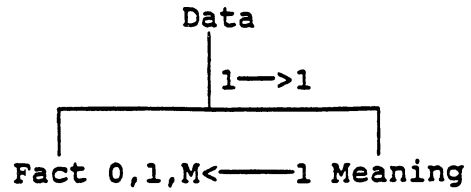


Figure 5, Definition of data (Appleton 1984)

Universe of discourse: the collection of meanings or objects about which information or knowledge is being expressed. These meanings or objects are interdependent. A meaning in this set is expressed in terms of other meanings. Without the universe of discourse, meanings cannot be interpreted. For example, the universe of discourse for a manufacturing system consists of machines, materials, parts, orders, handling devices, manufacturing processes and operations, etc. The meaning of these entities or notions and their use in expressing others in the set define the universe of discourse for a manufacturing system. The universe of discourse constitutes the total context of a particular system being defined.

Information: the aggregation of data needed for a specific purpose (Appleton, 1984). This definition implies that information should not be created if there is no purpose or demand for it. It also implies that information is made up

of at least one datum, if not several data. The definition of relations and/or operations on data generate information. In other words, data is manipulated or in some way related to other data to produce information. The determination of the need for information is the driving force of the definition of the relations and operations on data. For example, in the context of a manufacturing system, stock on hand (SOH) at the beginning of a period is a piece of information derived by the manipulation of the SOH datum from the previous period, quantity received datum and quantity consumed datum.

Knowledge: generated from a set of information through the application of experience, learning and logic processing. Grouping or clustering a set of related information around a concept or entity creates our knowledge about that entity or concept. The nature of these information bundles can be declarative or procedural. Heuristics provide additional information, relations and procedures about the problem domain beyond the information that is already available in the system. Application of heuristics on available information provides additional knowledge about the system. In a sense, heuristics are the application of experience and logic (in the form of a set of relations and operations) on information. Procedural knowledge is implicit in the sequence of operations of a procedure and is inseparable from the procedure that uses it (a set of operations). On the other hand, declarative knowledge can be explicitly

represented and stored in terms of symbol structures that are accessed by the procedures that use this knowledge (a set of relations). Our knowledge can be about objects, events, how to do things, what we know (meta-knowledge).

Intelligence: the ability to use knowledge to reason and learn. When a system is capable of representing the information and knowledge in an abstract way and has the ability to make inferences on what it knows and has the ability to learn (acquire more knowledge), that system is called an intelligent system. Since the definition of the term intelligence is somewhat vague, within the scope of this research the term is used as the ability to infer and create plans.

Following these basic definitions, the system sophistication levels for discrete part manufacturing systems can be formalized using the constructs presented in the following section.

System Sophistication Levels

Considering the definitions given in the previous section, five different levels of system sophistication are defined. These levels are utilized for distinct representation of physical and logical components of a system. The total system under study can be expressed as a collection of subsystems each of which is one of the five levels defined below. Figure 6 gives a visual description

of the levels along with the process necessary to move from one system level to the upper level.

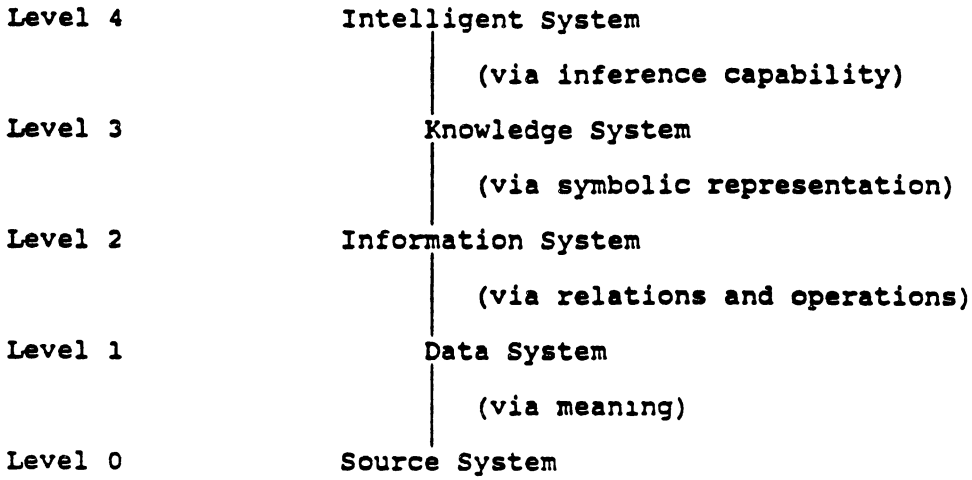


Figure 6, System Sophistication Levels

These levels, as related to discrete item manufacturing systems, can be formalized with the following set theoretic constructs.

Level 0 : Source System

This is the real physical system which provides the "facts" about the system. It can also be called a "Primitive System". At this level no meaning is provided for the observed facts. An outside observer perceives this system in terms of its interface channels, actions, physical inputs and outputs. The source system can be abstracted as follow:

$$SS = < P, read, F, T >$$

P is the representation for physical components, F

represents the set of observed facts, and **T** is the bounded real numbers set for time. The function **read** is a mapping function, explained following the next construct. An example of a **SS** is a machine tool.

$$P = < X, Y, C, OP >$$

where, **X** = set of physical inputs : This set represents the set of real world objects flowing into the source system. Each element of this set is represented by a unique identification. Some examples are work parts, tools, materials, fixtures, etc.

Y = set of physical outputs : This is the set of out flowing physical entities such as processed material, metalchips, scrap, etc. The elements of this set are elements of set **X** that are transformed into another form, and this transformation process that has been carried out by executing elements of set **OP** on elements of set **X** is reflected by changing the properties of elements of set **X**. The elements of set **Y** are represented in the same manner as are the elements of set **X**.

C = set of interface channels : This set consists of two disjoint subsets, C_{com} and C_{mat}. C_{com} represents the set of communication channels. The elements of this set are dedicated channels for external observation, communication and control. C_{mat} is the set of material flow

channels. These channels can be unidirectional or bidirectional. The elements of this set provide the soft and hard linkage of **SS** to the environment in which it resides.

OP = operations set. This set defines the set of physical operations that are performed in the physical system. At this level the links (logical relations) between interface channels, inputs, outputs and operations are not provided.

The function **read** of source system **SS** performs a mapping from one of the observation sets (**X**, **Y**, **C**) to the facts set **F**.

read: Z ---> F

where, the elements of set **Z** can be from set **X** or **Y** or **C**. Set **F** represents the set of observed facts during the simulation period. Thus, set **Z** can be expressed as the union of sets **X**, **Y** and **C**:

Z = X U Y U C

To give a better description of a source system, assume the following situation :

In a manufacturing system, an outside observer is watching a physical entity, i.e. a machine tool. The observer is looking at some physical entities that are flowing into the machine and some that are flowing out of the machine and some physical operations that are being performed. Define sets **X**, **Y**, and **OP** as follow:

X = {A101, J453, K234, T989}, and

Y = {J4532, A1013, K2349}

where, the elements of sets X and Y are unique identification symbols for different material (work parts). Similarly, assume that the operation set has the following operations expressed with unique identification:

OP = {D01, B34, F76}.

In addition, some form of communication is taking place on the communication channels. The observer is also supplied with some values on these channels. Define interface channels with the following set notation:

$C = \{01_{com}, 02_{com}, 03_{com}, 01_{mat}, 02_{mat}\}$

Notice that at this level the observer is not provided with the meaning of the observed facts (values and symbols). This is a dataless fact system. In other words, he/she is observing some facts, but unless the meanings of these facts are provided, the observer cannot make any interpretation of the facts. The time element is incorporated into this representation by defining a simulation period $T = \langle t_i, t_f \rangle$. t_i corresponds to the initial simulation time and t_f corresponds to the final simulation time. In other words, $\langle t_i, t_f \rangle$ is the time frame over which the system is simulated. When the meanings of these facts are supplied, the source system becomes a data system.

Level 1 : Data System

This is the source system equipped with the meaning of the observed facts. A Data System is represented with the following structure :

DS = < SS, U, assign, tmark, D, T >

where, **SS** = is the source system

U = is the universe of discourse for the source system. It is a set of predicates or names that are assigned and which give meaning to the facts. In a sense this set defines the context of the source system.

D = represents the data set. Each element, d, of the data set is an ordered 2-tuple of a fact and a meaning, d ϵ **D** and d = (f, u). The set **D** can be interpreted as the data base of the above source system. It is a collection of data values along with their names. Each name can assume multiple values corresponding to observations made at different instances of time while each value can assume a unique name. In most real world systems, centralized data bases are formed by unifying several **D** sets each of which correspond to a source system.

The function **assign** performs a mapping from the cartesian product of sets **F** and **U** to set **D**.

assign: F x U ---> D

It is the assign function which actually generates data from observed facts and defined meanings or names.

The time element that can be associated with each datum is used to represent the time that the datum is observed. The function **tmark** can be used for this purpose. **tmark** is a

function from data and time sets to data set D_t which represents temporal data (temporal data base). It can be used selectively (not for all elements of the data set) and assigns a distinct element of the continuous interval $<t_i, t_f>$ to a datum.

tmark: TD x T ---> D_t where, TD $\subset$ D

Since facts are no more than a collection of symbols without a meaning, time is associated with data rather than facts. Define sets U and D for the example given at level 0.

U = {part, operation, order, machine}

With the application of the **assign** function the following data set D can be obtained.

D = {$data_1$,$data_2$,$data_3$} = {(A101,part),
(D01,operation),(A1013,part)}

Now, if we want to associate time components for some elements of the data set such as operations, we obtain the following D_t set through the application of the **tmark** function.

D_t = { ($data_2$, t_1) }
= { (D01, operation), t_1) }

When the relationships and the operations between the elements of the data set are provided, the Data System becomes an information system. Notice that at this level, neither the interdependences between the elements of the universe of discourse (names and objects) nor the link between operations, inputs, outputs and interface channels are supplied.

Level 2 : Information System

This is a system that is evolved from a data system. A piece of information may be generated from a single datum or a set of data and each piece of information serves at a minimum one purpose, and may serve multiple purposes. The amount of information that can be generated from a set of data depends on the number of datums in the data set and the number of permissible relations and operations defined on that data set. An additional characteristic of the system at this level is its strong temporal component. Since the information is generated as required, the timely availability of information is crucial. The generated information can be immediately used and disposed of or stored. An Information System can be represented as follow:

IS = < DS, R, O, I, S, genInfo, look, action,
updateInternal, updateExternal, T >

where, **DS** = is a data system

R = is the set of relation definitions, which consists of two subsets, **R** = { **BR, NR** }. Notice that relations do not manipulate the data. Subset **BR** represents the set of Binary Relations. These are the relations between two datums or objects or datum-object pair. $BR_i \subset D \times D$ implies that binary relation i is an ordered pair of (d, d), (d, d) ϵ BR_i, where each of d is an ordered pair of (f, u). Thus, a binary relation

is an ordered 4-tuple of (f, u, f, u). Subset NR of set R represents the N-ary relations. These are the relations between n data items. In other words, $NR_j \subset D \times D \times \ldots \times D$ and ordered n-tuple $(d_1, d_2, \ldots, d_n)$ is an element of NR_j.

O = is the set of operation definitions which is made up of four subsets, O = { UO, BO, NO, CO }. Notice that operations do manipulate the data. UO is the set of unary operation definitions. These are the operations defined on a single datum. BO, NO and CO are the sets for binary, n-ary and conditional operation definitions, respectively.

I = is the information or internal state set. Each element of I is derived through the association of the elements of relations or operations sets to the data set of level 1, Data System. Each element of I, i, is an ordered 3-tuple of (a, d, t) where a ϵ R or a ϵ O and d ϵ D and t ϵ T. Depending on which subset of R or O a belongs to, d can be a single datum or a set of data and the association of a time component marking the creation or the last update time of the information (via tmark defined at level 1) provides the temporal relation for information. Thus, use of such multiple levels of time components keeps track of the availability of

data and information. The collection of elements of set I at any instant constitutes the internal state. This set represents the internal memory of the system. It is assumed that the system is capable of updating its internal memory in a single update operation and can distinguish any internal state from any other state.

S = is the set of external states expressed in terms of significant variables. This set represents the total state of the environment in which the system is operating. External state set is defined as a subset of the union of the internal states of all the entities. Filtering of the significant variables of each entity constitutes this set.

The function **genInfo** is a mapping from data set **D** and operations set **O** or relations set **R** to **I**.

genInfo: D x (R v O) x T ---> I

The symbol v designates the logical operator OR. Including time component t from set **T** provides for the timely generation of information.

Another function called **look** is utilized for partitioning of the external state.

look: S x C x T ---> E

Through the application of **look** on set **S** on proper interface channels, the information system obtains the variable values or object states that are necessary for its own operation.

E represents the set of external state variables observed through interface channels.

The function **action** stands for programmed decisions and selects the action that is going to be taken according to obtained internal and external states.

action: I x E x T ---> A

where, **A** represents the action set.

The internal state of the system is updated via the **updateInternal** function to reflect the effects of decisions selected by the **action** function.

updateInternal: A x I x T ---> I

This function defines the resulting new internal state each time a decision is made.

The effects of decisions made are reflected on the environment by the **updateExternal** function.

updateExternal: A x S x T ---> S

Application of this function may update the values of some variables in the environment or may affect the states of some other system entities. We can consider **updateInternal** and **updateExternal** functions as the actual implementers of the programmed decisions.

As can be seen, the information system carries more value than a data system. Single or successive application of relations or operations on data in a timely manner generates the information that is contained in information set **I**. Define arbitrary sets **R**, **O** and **I** for the example system given in level 1, Data System, to obtain an

information system.

R = {loaded, <, >, performed}

O = {+, -, /, *}

With the application of the **genInfo** function we can create an information set.

genInfo: $((d_1,d_3)$, -, t) ---> i_1

genInfo: $((d_2,d_1)$, performed, t) ---> i_2

I = $\{i_1, i_2\}$

= {(((A101,part,t_1),(A1013,part,t_3)), -, t),

(((D01,operation,t),(A101,part,t)), performed, t)}

The first information element i_1 corresponds to the time difference between the creation of parts called A101 and A1013, and the second information element i_2 expresses that operation D01 is performed on part A101.

The dynamic behavior of the information system can be defined with the following properties. They are used to represent how an information system moves in time from one state to another. Assume that at time t_0 an information system consists of the following:

$int(t_0)$ = **I** , internal state at time t_0

$ext(t_0)$ = **S** , external state at time t_0

$obs(t_0)$ = **look** (**S**, **C**, t_0) , observed state at time t_0

$act(t_0)$ = **action** ($int(t_0)$, $obs(t_0)$, t_0) , action taken at t_0 The system moves from t_{i-1} to t_i as follows:

$int(t_i)$ = **updateInternal** ($act(t_{i-1})$, $int(t_{i-1})$, t_{i-1})

$ext(t_i)$ = **updateExternal** ($act(t_{i-1})$, $ext(t_{i-1})$, t_{i-1})

$obs(t_i)$ = **look** ($ext(t_i)$)

$act(t_i)$ = **action** ($int(t_i)$, $obs(t_i)$, t_i)

This dynamic behavior can be explained with the following steps. First, a new internal state is obtained by using the **updateInternal** function which uses the previous internal state and the previous action selected. Secondly, an external state is generated by using the **updateExternal** function which uses actions selected at t_{i-1} and external state at t_{i-1}. Thirdly, an observed state is derived from the current observed state through the use of the **look** function. Finally, a programmed decision is made (an action is selected) utilizing the internal and observed states at time t_i.

Notice that at this level, the interrelationships among the elements of set U are not yet provided. When the information system becomes organized as a consequence of experience, learning, and more abstract representation, it becomes a knowledge system.

Level 3 : Knowledge System

This level is defined through the use of an Information System. There are two types of knowledge in a system, declarative and procedural. Declarative knowledge is the clustering of information about an entity or concept in a formal manner. Procedural knowledge on the other hand, is embedded into the procedures that are associated with entities or concepts and defines how specific actions are

executed. In a sense, knowledge gives contextual meaning to the information. A knowledge system can be defined as follow:

KS = < IS, SU, KB, genKnowledge, T >

where, **IS** = is the information system defined previously.

SU = is the structured universe of discourse. Up to this section universe of discourse was treated as a collection of names, but at this level **SU** defines the interdependences among the objects. Through the use of **SU** one can define an object in terms of other objects. When looked at from this level, **SU** defines the context and meaning for information. In an OOP environment, the inheritance tree can be considered as **SU**. In other words, SU defines the way an information set needs to be organized in order to be qualified as knowledge.

genKnowledge: **I x SU ---> KB** is a function which generates a knowledge base from a given information set and a structured universe. Any formal representation for **SU** which defines the interrelationships and interdependences among information components can be used for knowledge generation. **genKnowledge** maps the information set **I** of **IS** to Knowledge Base **KB** utilizing the structure and relations defined in **SU**.

KB = represents the Knowledge Base. It is the

organization and abstract representation of information about an entity, a concept or a procedure. This set consists of two subsets, DK and PK.

DK = represents the declarative knowledge. There are various representation schemes to express DK. Selection of one particular scheme affects the design of the reasoning mechanism that will be explained in the next chapter. Some of the most common schemes are predicate logic, semantic nets, frames, conceptual dependency, and scripts. These schemes range from syntactic to semantic spectrum of representations, where predicate logic is the most syntactic. A typical example of a DK is a Knowledge Base which contains a set of information (in AI literature, this set of information is traditionally called "set of facts". However, this definition of fact is different from the one provided at level 0).

For example, in predicate calculus form, the information about a machine performing a certain operation on a certain part can be expressed as

machineX (part(Y), operation(Z))

The collection of all "machineX" predicates defines the declarative knowledge about the parts and operations that are performed on a

machine called machineX. It relates parts to operations along with the machine that performs the operation.

PK = represents Procedural Knowledge. This type of knowledge is related to how specific actions are executed. It is a collection of operations that manipulate information. This type of knowledge is implicitly expressed in the sequence of operations. Therefore, the ordered collection of operations (which is a procedure) itself is called procedural knowledge. The representation scheme selected here must be in accordance with the declarative knowledge presentation scheme since **DK** and **PK** together constitute the knowledge base.

Following is an example of procedural knowledge as related to the DK example given above.

If: ∀ y ∈ Y and ∀ z ∈ Z [machineX(part(y), operation(z))
and machineX(status(idle))] ==>
perform(operation(z), part(y))

This example states that if there exist a part and an operation on machineX and machineX is idle then perform the operation on that part. In summary, **PK** is the set of permissible actions that can be searched and selected by an inference mechanism that is going to be defined in the next system level, Intelligent System.

As seen, the **KB** system structures the available information and expresses it in a symbolic representation scheme. The time component of the system can either be imbedded into a symbolic representation or can be handled explicitly. Here it is represented explicitly to handle temporal knowledge relations. Although not defined explicitly, a function similar to **tmark** defined at the Data System level can be used for this purpose. The sole purpose of this function is to keep track of the creation time for each piece of knowledge generated through the association of information set **I** and structured universe **SU**.

Define a knowledge system from an information system by defining **SU**, **genKnowledge** ,and **KB**. **SU** is the hierarchical tree-like structure which defines the role and hierarchical level of the information components in the information set I. It specifies the interdependences between the information elements of the information system (although it is possible to define a knowledge system over multiple information systems, this example assumes evolvement from a single information system to a knowledge system). Figure 7 graphically represents such a **SU**. Assume that **genKnowledge** is a function that takes the information components 1, 2, 3, 4, and 5 of the information system and generates a knowledge base in predicate calculus format according to the relationships defined in **SU**. Therefore, a knowledge base is a collection of declarative or procedural predicates that expresses the information about the system in an aggregated

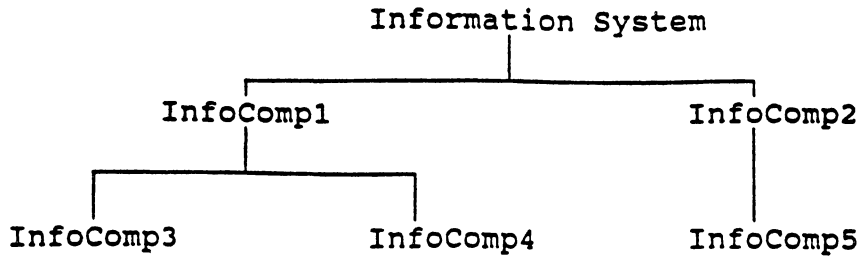


Figure 7, Information System

and organized form. Since practical implementation of the function **genKnowledge** is very difficult with the currently available programming languages, usually knowledge bases are explicitly provided. This fact reduces the sole purpose of function **genKnowledge** in this formalism into a function to provide consistency in the representation. Generation of knowledge directly from information by clustering related information around a given entity is a challenging task yet to be solved in the AI field. Actually, this is the process of translation from procedural programming languages designed to manipulate information to symbolic processing languages that are designed to process knowledge. This process is beyond the scope of this research.

A knowledge representation scheme which is a combination of frames and semantic nets can also be utilized to organize information. This type of knowledge representation actually forms the basis of Object Oriented Programming. In this view, instance and class variables are

used to represent the elements of set **I** which is basically an internal state set. The methods associated with the object can be considered as the procedural knowledge. Although pure OOP languages do not allow efficient implementation of inference mechanisms (reasoning or non-deterministic decision making mechanisms) for symbolic processing of the knowledge base, some OOP shells attached on top of AI languages like Lisp and Scheme bring the advantages of the object oriented paradigm into these languages. When a reasoning mechanism is associated with the **KB** the system becomes an intelligent system.

Level 4: Intelligent System

At this level, the system is capable of generating additional knowledge from the given knowledge base by utilizing a reasoning mechanism. The reasoning mechanism is used to process the knowledge base, thereby drawing useful and true inferences from it. In AI literature, reasoning and search through the knowledge base represents the same thing, especially when predicate calculus is used for knowledge representation. An intelligent system has the following components:

INS = < KS, EKB, EK, G, DC, perceive, decision,
updateIntKnowledge, updateExtKnowledge, T >

where, **KS** = Knowledge Base system from level 3.

EKB = is the set of external knowledge states. This is the set of knowledge about the environment in which the intelligent system is operating. This

set is analogous to the set **S** defined in Information System; here it contains knowledge about the total system, while there it contains information about the total system.

EK = a set of partitions of **EKB**. From all the knowledge available in **EKB**, only a certain subset is of interest to the intelligent system. Although the external knowledge base can be in different states within these subsets, it does not make any difference to the intelligent system and they all represent the same partition. Thus, an intelligent system can only discriminate between these partitions but not within partitions.

G = is the set of goals defined for the intelligent system. The intelligent system attempts to satisfy these goals by searching through its internal and external knowledge bases. These goals define the purpose of the search. Actually, goals represent the orders received from an upper level in the form of state descriptions.

DC = is the decision set. This set corresponds to the decisions selected via application of the **decision** function defined below. The elements of this set form an ordered collection of actions which gives non-programmed behavior to

the intelligent system defined at this level.

perceive: **EKB x C x T --> EK** is a function to partition external knowledge. This function is used to define subsets of external knowledge to be used for understanding the knowledge state of the environment. It is analogous to the **look** function defined at the Information System level which is used to access the state information for the total system.

decision: **KB x EK x G --> DC** is a function that decides on the actions that need to be performed to satisfy the goals (state descriptions) defined in set **G**. The function **decision** corresponds to an actual inference mechanism. There are two types of reasoning mechanisms that can be used. Forward reasoning starts with the initial knowledge (called facts in AI terminology) and searches until the goal is found (here, goal is the given predicate that is going to be searched through the knowledge base by unification and resolution. The inference mechanism will attempt to prove, evaluate to true, this predicate by utilizing back tracking). Backward reasoning on the other hand, starts with the goal and searches through the knowledge base to find a state that satisfies the goal. Forward and backward chaining are analogous to the

forward recursion and backward recursion of dynamic programming, respectively. In addition to these inference mechanisms that are applied using predicate calculus, some general purpose heuristic search procedures can also be utilized to support the reasoning mechanism. In summary, this function selects the action set from the knowledge base to satisfy the goals.

updateExtKnowledge: DC x EKB x T --> EKB is a function to update external knowledge according to the action taken. The purpose of this function is to reflect the changes which have occurred in the environment as a result of decisions made. Its role is similar to the **updateExternal** function defined at the Information System level.

updateIntKnowledge: DC x KB x T --> KB is the knowledge base update function. This function is used to update the state of the knowledge base once an action is performed. Selection of an action from the knowledge base makes certain facts true while making others false. This function is very much analogous to the **updateInternal** function defined at the Information System level and reflects the effects of selected actions on the internal knowledge status.

The dynamic behavior of an intelligent system can be explained with the followings properties. Assume that at time t_0 we have the followıng:

$int(t_0)$ = **KB**

$ext(t_0)$ = **EKB**

$obs(t_0)$ = **perceive** (**EKB**, **C**, t_0)

$act(t_0)$ = **decision** ($int(t_0)$, $obs(t_0)$, t_0)

The system moves through time with the following structures:

$int(t_i)$ = **updateIntKnowledge** ($act(t_{i-1})$, $int(t_{i-1})$, t_i)

$ext(t_i)$ = **updateExtKnowledge** ($act(t_{i-1})$, $ext(t_{i-1})$, t_i)

$obs(t_i)$ = **perceive** ($ext(t_i)$)

$act(t_i)$ = **decision** ($int(t_i)$, $obs(t_i)$, t_i)

As can be seen from the proposed formalism, the dynamıc behavior representations for information and intellıgent systems are almost ıdentical with some mınor differences. Although the dynamism of these two system levels are very much similar on the surface, they are actually far apart. First of all, the information system level uses information as opposed to knowledge used in the intelligent system level. The major implication of this fact is that the representation schemes used in these two levels are totally different. Secondly, since the decisions made at the information system level are programmed decisions, this system level can be implemented in a procedural like computer programming format. On the other hand, the intelligent system level requires a symbolic language like

computer programming format for efficient implementation of the inference mechanism. Therefore, the contents of sets defining these two system levels and the functions defined for each level are quite different. These issues are addressed in greater detail in the next chapter. Figure 8 summarizes the entire formalism.

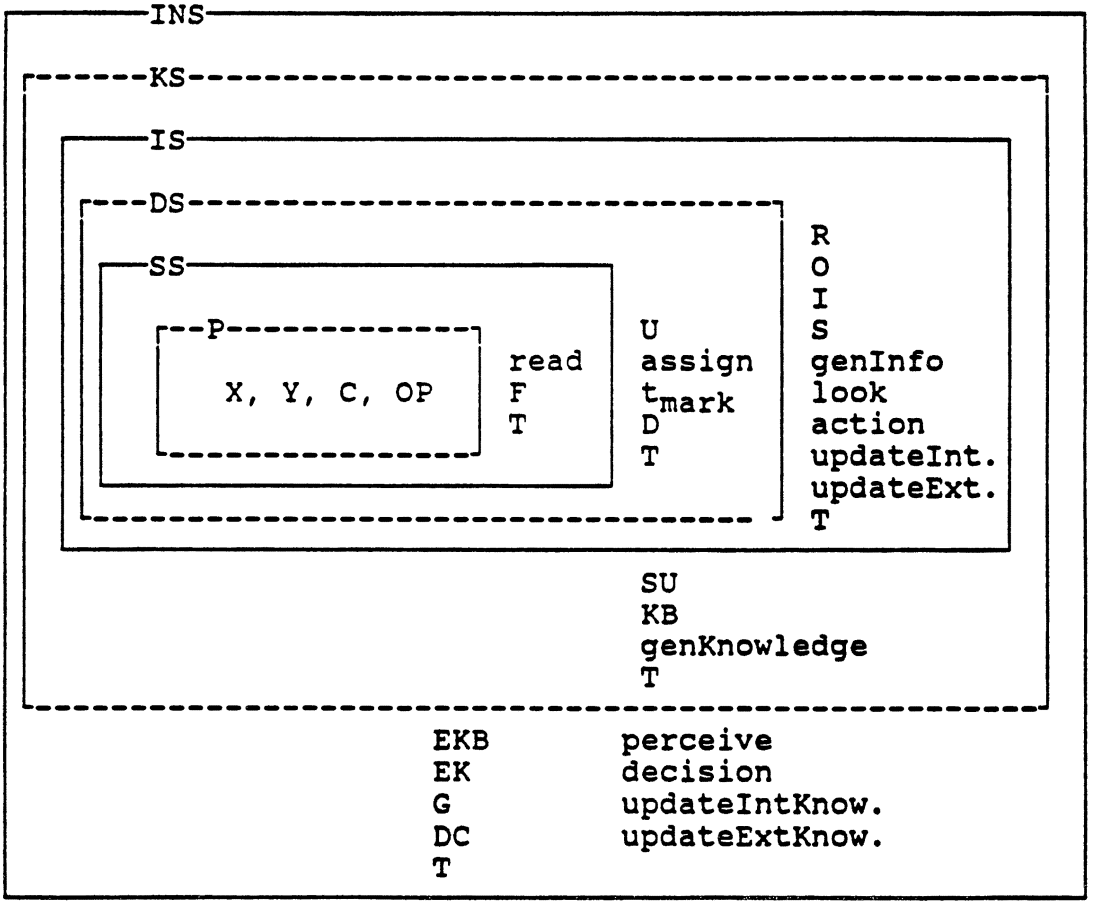


Figure 8, The Overall Structure of the Formalism

System Entities

In the formalism defined for discrete item manufacturing system simulation, the three major levels of system definition are source, information, and intelligent. Two system levels, data and knowledge, provide the necessary structures for the other three levels but do not play an active role. In other words, while source, information and intelligent systems define the behavioral representation along with some structural definitions, data and knowledge levels basically define structural representations. On the other hand, while system dynamism (moving in time from one state to another) is expressed and controlled in the information and intelligent system levels, the source system level provides the physical structures for the execution of real physical activities. In reality, these physical activities are direct results of programmed and non-programmed decisions made at the information and intelligent system levels, respectively. The main functions of the data and knowledge levels are to facilitate the decision making processes with their formal constructs. In a sense, these two levels lay down the background for proper operation of the information and intelligent system levels.

Once the above formal structural and behavioral representations for a system have been provided, the interactions between physical and decision making (intelligent) elements as well as interactions among

decision making elements can be established through the use of control, communication and material flow relations. The interface channel set C which includes material flow channels C_{mat} and communication channels C_{com} is the standard structure for this purpose. Notice that all the entities in the system are initially stand alone objects. When these entities are bound together with data, information and knowledge, structural and behavioral representations at the total system level can be easily achieved. Since the interface channels are uniform structures throughout the system and all types of flows are initiated at (or decided on) sending ends and interpreted at receiving ends, defining interrelationships is relatively straight forward in this form of representation. Additionally, this formal representation is very similar to the way relationships and interactions are established in complex real dynamic systems where timely information is crucial and intelligent entities are the key players.

Since the main emphasis in this research is discrete parts manufacturing systems, a taxonomy is defined for manufacturing system entities. Entities defined in this taxonomy are the main classes of a discrete item manufacturing system. First of all, as expressed in the formal definitions presented in the previous section, a clear distinction is made between the system entities according to their level of system sophistication. Since the system dynamism is represented in two distinct levels of

system sophistication, namely, information and intelligent system levels, the entities are divided into two main categories as "Decision Making Entities (DME)" (intelligent agents) and "Information or Data Driven Physical Entities (DDPE)" with some control structures attached. This grouping allows explicit abstraction of cognitive activities that take place in manufacturing systems. In reality, two major types of activities are performed in purposeful systems, physical and cognitive activities. The formalism defined provides the basic structures for distinct representation of these two types of activities as two disjoint sets with heavy interactions between them.

Decision Making Entities

These are the basic building blocks of the planning and control system. The entities at this level can be represented with the properties of intelligent system level constructs of the formalism. Based on their internal knowledge and what they know about the rest of the system, these entities make non-programmed decisions. The non-programmed decisions are the decisions based on the current knowledge state of a particular intelligent entity. In some literature, decisions of this type are called non-deterministic because the decisions made are the results of the state in time and show differences as the system moves in time. The entities in this group search through their knowledge bases and select a set of actions which best

satisfies the given goal. In a sense, the non-deterministic decision making process is planning and problem solving in a restricted domain. The proposed information and knowledge processing schemes as well as information and goal decomposition methods for decision making entities and physical entities will be discussed in the following chapters.

For the purpose of defining a meaningful taxonomy, three different levels of decision making entities are defined utilizing the research done at MIT (Chryssolouris and Gruenig, 1988).

1 - Entities that deal with the planning and control activities at the product level. The cognitive activities of these entities correspond to the conventional strategic level activities. For the sake of simplicity, classical strategic level activities such as acquisition of new facilities, new markets and financial planning and control are not included in this study. These activities require a lot of expertise in finance and economics and depend on the particular industry and the economic system in which the manufacturing system is operating. The entities at this level prepare relatively long term plans in terms of what to produce when. Thus, the main output of this level is a master production schedule like plan. In their activities, this entity or group of entities utilize the current state of the whole manufacturing system in a filtered form, the system work load for the period of interest and the customer

demand (expressed in terms of quantity and due date) for various types of products. These are the only aspects of interest at this level.

2 - This second level of decision making entities plan and control the major jobs that need to be carried out and synchronized in order to conform to the plan prepared by the above level of decision making entities. Once the product orders (or master production schedule) are decided upon, entities of this level plan and control the necessary activities at the shop (or departmental) level. Based on product structure, product orders are broken down into job orders that are under the responsibility of the Decision Making Entities at this level. The main types of knowledge utilized by these entities are the product orders generated at the upper level and the current state and status of each shop (in terms of load and resources) planned and controlled by each entity. Entities at this level create their individual job plans (when to do which job) aimed at the satisfaction of product orders.

3 - The third level of decision making entities constitutes the planning and control activities at the work center or cell level. At this level, the job orders, derived from product orders and generated by second level decision entities, are further broken down into task orders. The main objective of the entities of this level is to synchronize a set of tasks in order to conform to various job orders. The task orders are mainly derived from the

process plans and routing information of the products. This is the lowest level in which intelligent behavior (or non-programed decision making) takes place in the hierarchy. Although task orders are further broken down into activity orders, it is assumed that each task order is made up of a finite set of ordered activities. Activity orders correspond to physical activities that need to be performed in order to satisfy task orders. Since the physical activities take place at the lowest hierarchical level and executed in a predetermined sequence to carry out each task order, there is no need for real planning at the activity order level. Additionally, since the activities are performed in a deterministic manner (we know exactly what to do after each activity) a deterministic control scheme is proposed for real physical activities. Figure 9 gives a pictorial representation of the Decision Making Entity (DME) taxonomy.

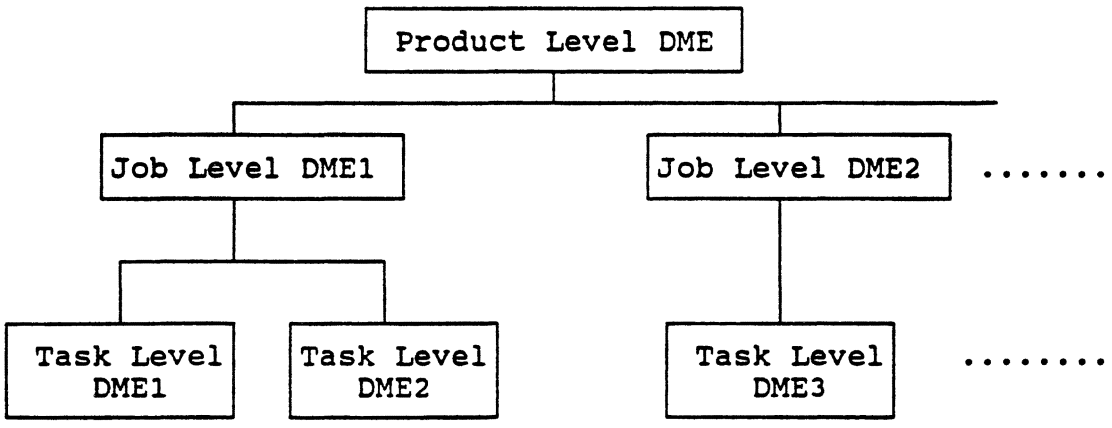


Figure 9, Decision Making Entity Taxonomy

Data Driven Physical Entities

These entities form the bottom level or the foundation of the manufacturing system. Their activities correspond to real physical activities that are taking place on the shop floor and are direct results of decisions made by decision making entities. We can consider these entities as the real implementers or executors of the decisions. As briefly mentioned at the end of the previous section, activity orders are carried out by these entities. In real systems, they are the individual machine tools and other physical devices. Planning at this level of system entities coincides with the deterministic or programmed ordering of activities. For example, when a task is assigned to a machine tool, the set of physical activities to satisfy the task are the following:

- pull the item that is addressed in the task order from the input queue,
- set-up the machine (if necessary),
- load the material (work part) on the machine,
- perform the requested operation on the material,
- perform on-line inspection (if appropriate),
- unload the material,
- put the material in the output queue.

As can be seen, the sequence of operations is deterministic and we cannot perform one activity without performing the preceding one. The control modules attached to this machine tool or device also takes care of the logic checks such as

existence of material and instructions to perform the processing operations, etc. In addition to these logic checks, the control structure handles the bookkeeping operations for this device.

Another major function of the control module attached to the entities of this type is the monitoring of activities and exception handling. Exceptions are the events or occurrences that force individual entities to deviate from their deterministic activity sequence cycle. Some examples of exceptions are listed below:

- machine breakdowns
- set-up operations
- material shortage
- information shortage
- queue capacity problems
- tooling unavailable
- machine operator absent

When an exception other than set-up is encountered, the entities of this type immediately inform their supervising Decision Making Entities at the task level with appropriate messages. At the same time, they take care of the basic actions that need to be performed as part of exception handling, i.e. locating material back to the input queue or aside when a breakdown has occurred. In summary, entities of this type constitute the building blocks for the physical system and highly resembles the performance level of Nadoli and Biegel (1989). Figure 10 gives an example taxonomy for

physical system entities.

A major implication of the hierarchy defined for Decision Making Entities is its ability to allow natural analysis of information components. The information elements in a manufacturing system are the main driving forces of the physical and logical systems. Information is the binding factor between physical and logical (planning and control) systems as well as within the logical system. Each Decision Making Entity deals with the type of information that is appropriate to its own level. The top level Decision Making Entities are concerned with the customer orders and current and future state of the facility at the shop or departmental level. These entities generate

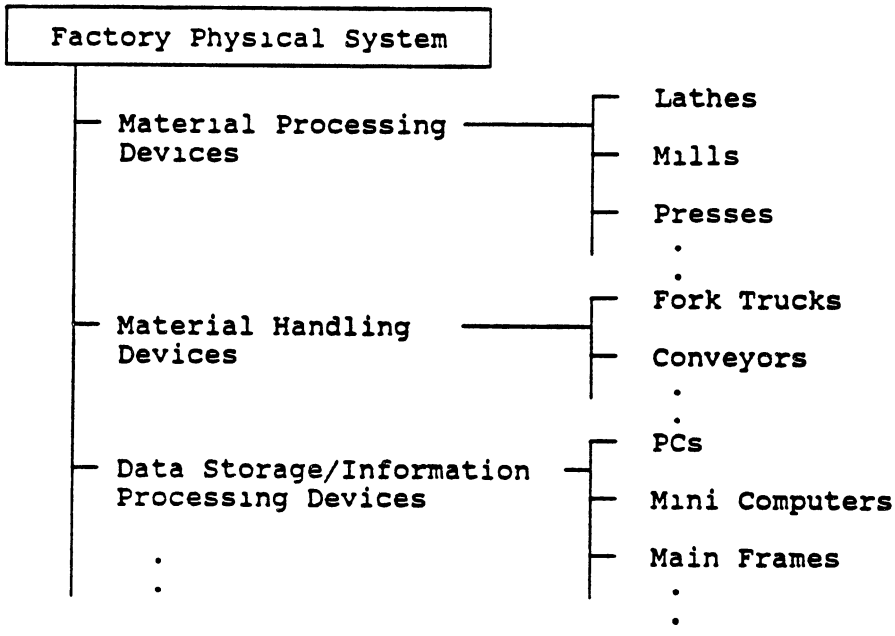


Figure 10, Physical System Taxonomy

product orders in a timely manner as a result of their cognitive activities. The entities of the next level are only interested in job orders derived from product orders and deal with the restricted domain of their own assigned shop. Besides product orders, their main information requirements are the knowledge on the current and future status of the work centers that are operating in their department. The third level of Decision Making Entities deals with the task orders created from job orders and assign tasks to individual devices of their own work center. The status of its assigned devices are the major concern of this level of entities. Finally, at the bottom, the task orders are accepted by devices and transformed into a set of ordered activities.

As can be seen, each intelligent entity accepts orders from its superior entity, performs its own cognitive process by deciding on what to do when and initiates proper orders to its own subordinates. Thus, each planning and control problem is solved in its own local domain by the most knowledgable entity about that area. This whole process is pictured in Figure 11.

The next chapter focuses on a proposed information and knowledge processing scheme for the simulation of the manufacturing systems whose formal representation is presented in this chapter.

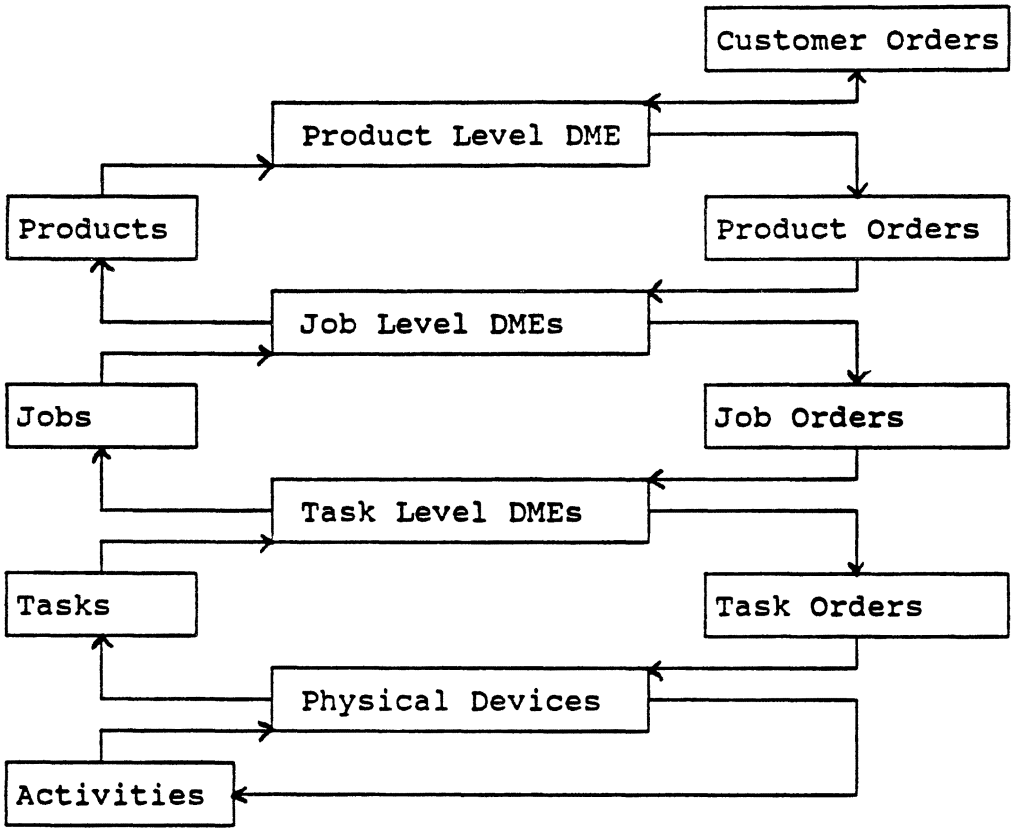


Figure 11, Decision Making Entities of a Manufacturing System

CHAPTER VI

INFORMATION AND KNOWLEDGE PROCESSING

This chapter addresses the issues related to processing of information and knowledge. Since a clear distinction is made between these two terms, the chapter is organized around two main sections namely, information processing and knowledge processing. These two closely related yet quite different sections correspond to actual decision making processes at the information and intelligent system levels of the developed formalism.

Introduction

In the previous chapter, when the formal representations for system entities were presented, two functions named **action** and **decision** were used in the information and intelligent levels, respectively. At the information system level, function **action** takes the given internal and external states and maps them into an action that is going to be implemented by two distinct update functions. Similarly, the function **decision** of the intelligent level takes the local (or internal) knowledge base along with the perceived external knowledge base and the goals defined for that particular intelligent entity and

maps them into a set of actions. What is not provided in the formalism is the way the action or the set of actions is selected. Therefore, this chapter gives a detailed treatment of the methods that define how those actions are selected at both the information and intelligent system levels. Before proceeding further, even though they are intuitively well understood, the concepts of state and change require some explanation.

States and Actions

The notion of state is central to the simulation of the physical world. It can be defined as a snapshot of the real world at a given instant in time. The number of states that a physical world can be in is a function of the number of entities and relationships defined in the system. For large systems with many relationships, the number of states can easily be very large. But one key concept, especially very useful in simulation modeling, is that conceptualization of states need not be unique. In other words, even though a system may be in several different states, as a result of our state definitions it could represent the same state in our conceptualization. The main value of the state notion is that it lets us describe the changing world. We can define a set of terms called state designators to denote specific states. But the simplest way of describing a state is to use a single relation for each piece of information about the state.

While the main concept behind the state notion is to represent stability, the concept behind the notion of action is to represent change. The state of the world stays in one state until the execution of an action that takes the world to a new state. Actions can also be expressed as objects in the universe of discourse. Action designators specify the names of the actions and the entities involved. A set of functions called "update functions" can be used to represent the effects of actions. These functions map an action and a state into a new state that results from the execution of the action.

Information Processing

In the literature, the term information processing is used for a wide spectrum of intentions, but in this research the term "Information Processing" refers to selecting an action by utilizing given external and internal states and implementing this selected action by making appropriate changes in internal and external states that occur as a result of performing the action. Thus, the main topic presented in this section is the detailed explanation of the operation of function **action**.

The information system level of the formalism uses information elements (the elements of set I and S) as the fuel of its operation. Information elements are the actual descriptors of state and change at this system level. Since the function **action** is based on deterministic control

concepts, there is no intelligent behavior at the information system level. In the context of discrete part manufacturing systems, the information system level corresponds to physical level manufacturing system entities with control structures attached. The behavior of these entities are state dependent. In other words, every state reached is a function of the previous state and the current state defines the permissible actions and resulting states from the execution of these actions. The definition of states depends on the nature of the physical entity represented and the control mechanism employed. For example, the following four states can be defined for an individual machine tool of a manufacturing system.

- idle
- busy
- broken
- in-repair

Figure 12 graphically represents state transitions for such a machine tool. In Figure 12, the machine tool starts the simulation with an idle state (it is also possible to start a simulation from any state). When a task order is assigned to the machine, it changes its state to busy by updating its internal state and at the same time updates the external state (the state of the total system). These state changes are implemented by changing the values of the variables that define the states. If there is no task order assigned, the machine stays in the idle state. When the machine is in the busy state, there are three things that can happen.

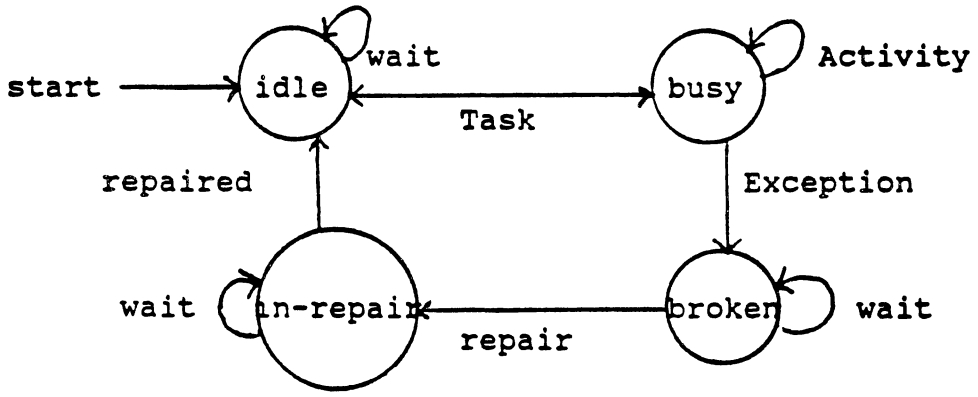


Figure 12, State Transition Diagram of a Machine Tool

First, if there exist other task orders assigned to the machine during this busy state, the machine object keeps satisfying them by staying in busy state. Second, if there are no more job orders assigned at the end of the busy period, the machine object changes its state to idle. Third, during the busy state the machine may transfer into a breakdown state. Once the breakdown state is reached, the machine can only make transition into in-repair state or it may stay in broken state until the conditions for that transition are satisfied. When the machine is in in-repair state, it can only make transition to idle state. This closed state transition cycle is repeated as long as there exist task orders assigned, or until the termination time for the simulation. The above Figure only shows the defined states and transitions of a machine tool object; it does not

include the control mechanism associated with the machine. It is the control structure (we can also call it a control object) where actual information processing takes place. The function of the control object is to handle logical condition checks on state transitions, select the appropriate transition and to actually implement the state transition by changing the values of variables that define the states. Although the number of actual states for an information level system object can be very large and the transitions can be very complex in the real world, we can define states and transitions in a partitioned way that best fits our modeling purpose. Figure 13 gives a graphical representation of the control object's typical operation.

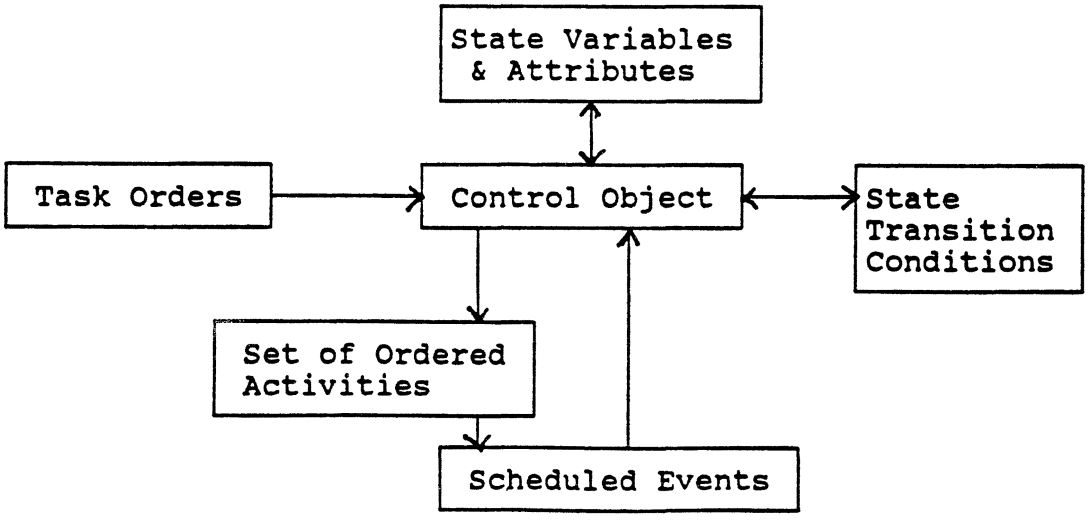


Figure 13, Control Object Operation

In Figure 13, the control object receives task orders from the task order level decision making entity. Task orders are the major structures for linking decision making entities to information system level entities. Upon receiving task orders, the control object checks on the state transition conditions by accessing necessary state variables. If the state transition conditions are satisfied the state variables and attributes are updated in order to reflect the changes. Then, the control object accesses the set of ordered activities defined to satisfy the task orders. It performs these activities by scheduling events into an event calendar to implement the passage of time for the execution of activities. These scheduled events are later handled again by the control object. Dynamic behavior of the information level entities are created by the pseudo functions defined in the formalism. These functions, namely, $int(t_1)$, $ext(t_i)$, $obs(t_i)$, and $act(t_i)$ are defined to move an information level entity in time and they coincide with the time advancement mechanism of the simulation. Therefore, the control object triggers the execution of actual physical events and at the same time monitors the operation of the physical world. Another important function of a control object is to report (on-line or periodic) to its task level decision making entity. These reports are the actual feed-back information utilized by decision making entities in their cognitive activities.

Besides the above routine handling of task orders, the

control object takes care of exceptions. As defined in Chapter V, exceptions are the type of events that cause the control object to deviate from its routine task. Several examples of exception events are given in the previous chapter. The breakdown event, which is included in Figure 12 by the state called broken, is a good example of an exception. When an exception event is encountered, the control object handles this exception by specifying what to do. The problems related to the global effects of exceptions are handled at decision making entity levels namely, the task order level decision making entities. In case of an exception event, the major function of the control object is to inform its superior task order level decision making entity and to update the information level entity's internal state to reflect the effects of the exception event occurrence. This process implies that every control object has a set of procedures that defines the actions needed to be performed in case of an exception. Each element of this set corresponds to a certain type of exception event. This aspect of the control object can be explained with Figure 14. In Figure 14, regular events are the events like completion of service, arrival of material, arrival of task order, etc.

As can be noticed, the control is represented explicitly in the implementation of information level entities. This is fundamentally different from the conventional implementation of the similar entities where

control is embedded into the entity structure. There are certain advantages associated with the above implementation.

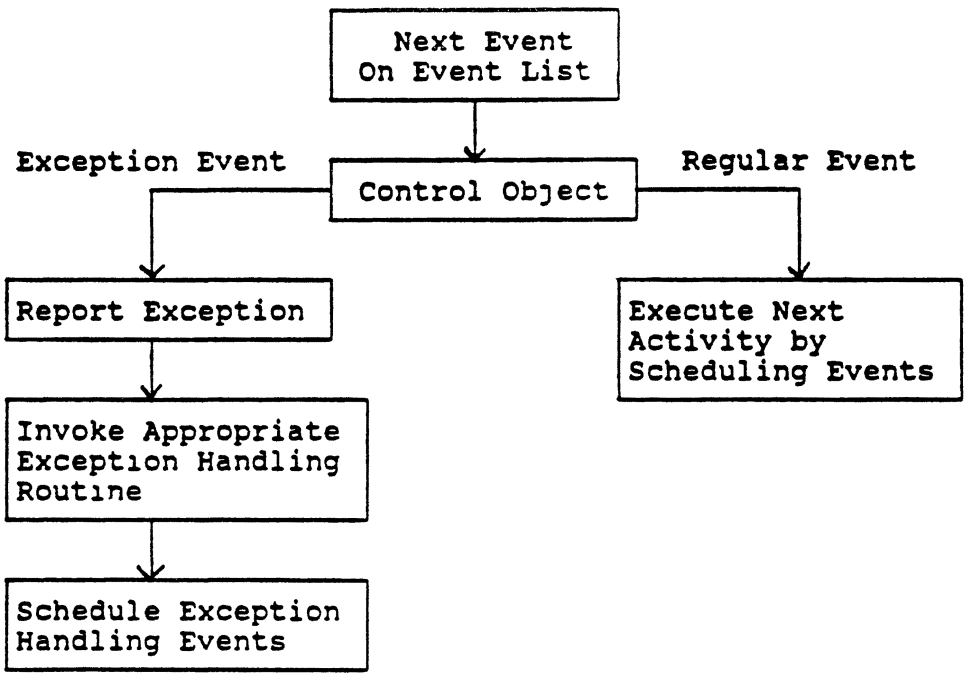


Figure 14, Exception Events and Control Object

First of all, explicit representation of control provides better flexibility for defining governing rules of this type of entity. It localizes the changes related to the control structure employed and makes modifications easier. Secondly, it attempts to take full advantage of the inheritance mechanism provided in OOP languages. A generic control object outlines the basic structure and behavior for the most common type of control that is usually employed in a system. The subclasses of this generic control define more specific types of control objects that are needed.

After explaining the information processing aspect of the developed framework within this section, the next section illustrates the processing of knowledge.

Knowledge Processing

Within the context of this research and the formalism developed, the processing of knowledge refers to making decisions at intelligent (or decision making) entities by selecting one of the alternative actions. This process corresponds to the operation of the function **decision**. As a prerequisite to knowledge processing, the decision making entity must face a decision problem. Once the entity's internal knowledge state and the state knowledge about the rest of the system is assessed, the next step is the determination of the possible courses of action that can be taken to move into a desirable state in the light of the decision problem's objective. It is the following stage where actual knowledge processing takes place through the use of the **decision** function. In this stage, each alternative action is evaluated to see the resulting state if that action would have been selected. Following the evaluation of each alternative is a comparison process to see how well the resulting state conforms to the desired state. The desired state is usually specified in the goal of the decision problem on hand at that instance of time.

The process described above is a complex procedure that

requires a search process through the knowledge base. Since manufacturing system simulation is actually the simulation of decision making processes, the physical entities of the system act as passive elements, as compared to decision making elements, whose only function is to implement the non-programmed decisions made at upper levels. Contrary to the conventional way of visualizing simulation in which physical entities of the system are the major players, this approach focuses on decision making entities as the key elements. A unique characteristic of the knowledge processing scheme proposed in this research is that all decision problems are perceived as planning problems. This is due to the time component associated with information and knowledge in the formalism developed. Other systems developed in the direction of intelligent system simulation assume perfect information, that is whatever information an entity needs is immediately available. Furthermore, in most of the systems developed or proposed, decision making processes consume no time, which is not realistic. Although they are purely logical, like physical activities, cognitive activities take some time due to the complex process of decision making. Another way of stating this fact is that decisions are not made instantaneously when a decision making problem is encountered. Selection and evaluation of alternative courses of action consume some time. This aspect of intelligent system simulation is addressed, well discussed, and included in the system developed at Los

Alamos (Burns and Morgeson, 1988). Also in many cases, a decision making entity has to wait for a piece of information that is not currently available, e.g., waiting for a status report from a subordinate level before allocating resources to a set of tasks or jobs. Another major characteristic of other studies which highlights this effort's important aspect is the exclusion of planning in intelligent entities. When planning is not the major task of an intelligent entity, that entity functions as a reactive entity rather than an active entity. In other words, in other systems intelligent entities only deal with immediate decision problems and when a decision is made it is instantaneously implemented. In reality, this is not the way intelligent entities of a manufacturing system function. Planning for the future is the major activity of intelligent entities of a purposeful manufacturing system. This property of other systems takes away the major portion of the behavior that has been traditionally called intelligent behavior.

The planning function of decision making entities is the main capacity of this type of entity. Planning involves more than making intelligent selections among possible alternative actions. It includes time ordering of selected actions along with synchronizing activities with the activities of other entities. This synchronization process also includes proper communication with the rest of the system. Since the representation scheme or the fuel of

decision making entities is implied as knowledge rather than information for the function named **decision**, the next section addresses the issues related to the knowledge bases utilized for proper operation of these entities.

Knowledge Base

The knowledge presented in the form of predicates at Decision Making Entities must be processed in order to carry out planning activities. The body of declarative and procedural knowledge about the total system and its operation is contained in the knowledge base. In the literature, several different ways of structuring and organizing knowledge bases for system simulation have been proposed (Radiya, 1987, Shannon, 1988, Reddy, 1986, Castillo, 1989). In some of the proposed systems, the whole body of system knowledge is accommodated in a single knowledge base in terms of production rules. This knowledge base is searched to infer new facts every time a non-programmed decision problem is encountered. Realizing the inefficiency of this process, a group of researchers from McDonnell Douglas Astronautics at Kennedy Space Center (Castillo et al., 1989) propose a scheme called HSKB (Hierarchical Segmented Knowledge Bases). After highlighting the drawbacks of new knowledge based simulation systems such as inefficiency of searching through large rule bases and hard coding of IF-THEN logic to objects, they

define HSKB as a simplified means for performing embedded reasoning during simulation. This study utilizes a "script driven simulation" paradigm. Scripts contain temporally ordered activities and are analogous to plans.

In this development effort, knowledge bases are perceived as a collection of rules about possible sets of actions and each of these knowledge bases is specific to a certain type of problem. These domain specific knowledge bases are attached to decision making objects. Therefore, each object is capable of dealing with certain types of problems that are typically encountered in their cognitive activities. Since the regular decision making problem in the proposed system is defined as a planning problem, search through knowledge bases corresponds to the selection of a sequence of activities to satisfy the goal stated in the problem. The objective or the goal of the problem is defined in terms of a specific state in which the decision making entity would like to be at some time in the future. These goals are actually the orders received from a superior entity. With its regular planning activities the intelligent entity tries to satisfy these orders. The time component acts as a constraint for temporal sequencing of activities. Once the current state and the work load of the entities that are controlled (or are under the responsibility of this entity) are assessed through the functions defined in the formalism, the next step is the interpretation of the problem content to understand the

problem type faced. The problem types can be regular planning problems expressed as resource time allocation problems (in case of alternative resources, they also include resource selection) or various types of exception handling. As soon as the problem type is determined and current knowledge on the state is available, search through the problem knowledge base starts. Since this process corresponds to thinking defined in terms of selection and evaluation of alternatives in the real world, a time delay is associated with it. This implies that the decision will not be available for implementation until some time in the future. Notice that the resulting decision is not a single action. Rather, it is expressed as an ordered set of activities. Figure 15 summarizes the operation of the proposed knowledge processing mechanism for regular planning activities.

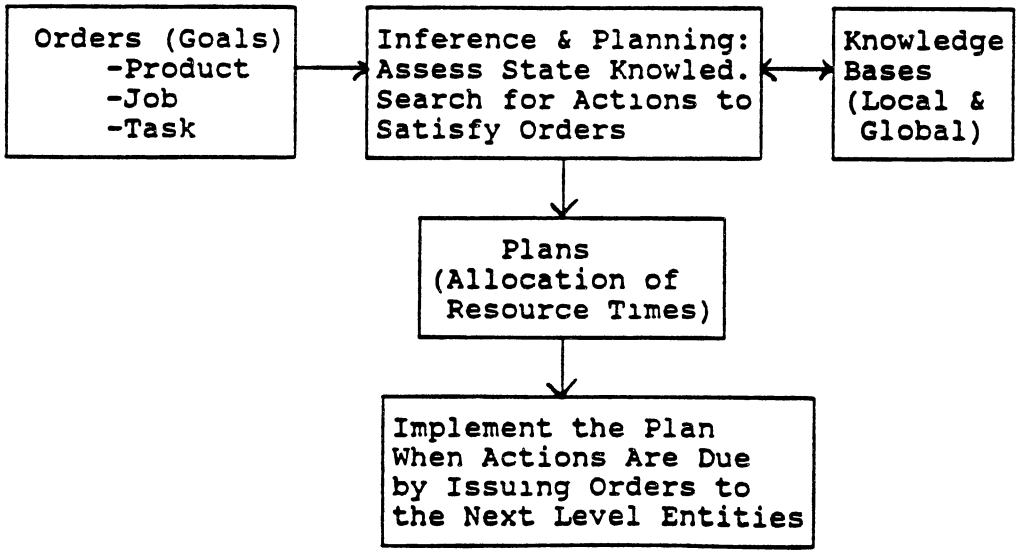


Figure 15, Knowledge Processing

For the processing scheme defined here, a planning method called "Goal Regression" is utilized (Genesereth and Nilsson, 1987). The next section gives a detailed description of this method. As stated before, non-programmed decision making is defined as a planning activity in this approach. The planning in a manufacturing system corresponds to the allocation of resources in the form of machine, work center, shop, and department times to the orders received from the supervising levels. Therefore, within the context of this research, the planning problem can be viewed as a dynamic resource allocation problem. The changes in the states of orders and resources affect our allocation process. Considering this perception, for the cognitive activities of the decision making entities outlined in the previous section, some state descriptors are defined. Two main entities involved in the planning process are the order objects and resource objects. Depending on which level planning activity is taking place, order objects can be an instance of product, job, or task order with attached due dates and estimated processing times. Similarly, a resource can be a shop, a work center, or a machine. Although each of these order and resource objects has individual characteristics, they can be one of the defined general states. An order o_i can be either in an **Assigned** or **Unassigned** state expressed with the following state descriptors:

Assigned (o_i, r_j, t_s)

Unassigned (o_i)

where, r_j is the resource j and t_s is the starting time. Likewise, a resource r_j can be in one of the two states:

Available (r_j, t_b, t_f)

NotAvailable (r_j, t_b, t_f)

t_b and t_f correspond to beginning and ending times in the formulation. In addition to these state designators, two more are defined on orders to access the due date and estimated processing time information:

$$\text{duedate}(o_i)$$

$$\text{estPt}(o_i)$$

In conjunction with the state designators defined up to now, the following action designators are defined to describe the actions that cause changes in the states:

Assign (o_i, r_j, t_s)

Cancel (o_i, r_j, t_s)

The first one conceptualizes assigning an order to a resource and the second one represents the cancellation of an assignment. Therefore, the problem of time allocation is solved by assigning resources to orders one by one and backtracking as soon as it is realized that the goal is unachievable (one of the order due dates cannot be met). The backtracking mechanism provides for the exploration of another sequence of orders that may or may not satisfy the due dates. When the first sequence of orders that satisfy the due dates is obtained, that sequence becomes the

operation plan and communicated to the next level as an order. A conflict resolution strategy that gives priority to **assign** actions can be employed for the process described. Conflict resolution refers to making a selection between candidate actions when more than one is available. This brief explanation of state and action sets defined for the dynamic resource allocation problem sets the stage for the introduction of the Goal Regression planning strategy.

Goal Regression

Although there are various planning schemes proposed in the AI literature, the one called goal regression is one of the most simple yet powerful planning strategies. Especially in situations where each action is characterized in a simple form, goal regression works very well. In this scheme, each action instance is characterized by a set of prerequisites, a set of positive effects, and a set of negative effects. The prerequisites **Pre(a)** of an action **a** are the conditions that must be true in order for **a** to have the desired effect. The positive effects **Add(a)** are the conditions that become true after the action is executed. The negative effects **Del(a)** are the conditions that become false as a result of executing **a**. Contrary to conventional rule based systems, goal regression provides for deletion of some knowledge elements (state descriptors) by making them false. This fact actually provides the nonmonotonic aspect

for knowledge processing. As an example of this transformation, consider the action called **Cancel** that is defined in the previous section. This action was expressed with the operator

$$\textbf{Cancel}\ (o_i,\ r_j,\ t_s)$$

and can be defined in terms of state designators that are also defined in the preceding section. First of all, to consider such a cancellation, order i must already be assigned on resource j. This condition is expressed with the state designator **Assigned** $(o_i,\ r_j,\ t_s)$. Furthermore, order i must be meeting the due date. This can be checked with the formula $(t_s+\text{estPt}(o_i) \leq \text{duedate}(o_i))$. These two conditions are the prerequisites of a **Cancel** action.

$$\text{Pre}(\textbf{Cancel}(o_i,r,t_s)) = \{\ \textbf{Assigned}(o_i,r,t_s),\ (t_s+\text{estPt}(o_i) \leq \text{duedate}(o_i))\ \}$$

The positive effects of the Cancel action are stated with the following set. Cancellation of an order makes state designator Unassigned(o_i) true and makes the assigned time slice of the resource free.

$$\text{Add}(\textbf{Cancel}(o_i,r_j,t_s)) = \{\ \textbf{Unassigned}(o_i),\ \textbf{Available}(r,t_s,t_s+\text{estPt}(o_i))\ \}$$

There are two negative effects of the action which make state designators **Assigned**(o_i,r,t_s) and **UnAvailable**$(r,t_s,t_s+\text{estPt}(o_i))$ false.

$$\text{Del}(\textbf{Cancel}(o_i,r_j,t_s)) = \{\ \textbf{Assigned}(o_i,r,t_s),\ \textbf{UnAvailable}(r,t_s,t_s+\text{estPt}(o_i))\ \}$$

In this example, we may define the goal set to be a state set, such that time slice t_s to t_f of resource r_j is clear.

$$\{ \textbf{Available}(r_j, t_s, t_f) \}$$

The main philosophy of goal regression is explained in (Genesereth and Nilsson, 1987) as follows:

> "The basic step in goal regression is the reduction of one goal to a subgoal on the basis of an action description. This reduction must have the property that performing the described action in a state satisfying the subgoal will produce a state satisfying the goal. Given the preceding definitions, we can see that the subgoal Reg(q, a) resulting from the regression of q through the action a consists of the prerequisites of a together with the members in q that are not among the positive effects of a. Furthermore, for the action to work, there must be no overlap between the negative effects of the action and the conditions in the goal."

These properties are expressed with the following formulation.

$$(q \cap \text{Del}(a)) = \{\} \Longrightarrow \text{Reg}(q, a) = \text{Pre}(a) \cup (q - \text{Add}(a))$$

The relation plan defined in goal regression is a 3-tuple made up of a goal set, a state, and an action sequence. This relation is true iff the resulting state, attained by executing the action sequence when we are in the given state, is in the goal set.

$$\text{Plan}(q, s, l) \Longleftrightarrow T(q, \text{Do}(l, s))$$

where, q is the goal set, s is the given state, and l is the action sequence. The Do function used above represents the implementation of the action and corresponds to the

generalized form of **updateIntKnowledge** and **updateExtKnowledge** functions of the formalism.

The definition of goal regression can be used to define the conditions under which an action sequence is a plan. First of all, an empty action sequence is a plan for goal set q in state s if s satisfies the elements of q.

$$T(q, s) \Longrightarrow Plan(q, s, ())$$

The action sequence a.1 (.1 indicates a sequence) is a plan for goal set q if it satisfies the following two conditions:

1- a is an action and the positive effects of a include an element of q.

2- 1 is a plan that achieves the goal set obtained by regressing q through a.

These two conditions can be expressed with the following formulation.

$$(q \cap Add(a)) \neq () \wedge Plan(Reg(q, a), s, 1) \Longrightarrow Plan(q,s,a.1)$$

In summary, goal regression is the problem of finding a series of actions α such that $Plan(\phi, \sigma, \alpha)$ is true, where ϕ is the goal descriptor and σ is the initial state descriptor. Fundamentally, the Goal Regression planning scheme is similar to the way rule based inference engines operate, but there are some significant differences. Firstly, rule based inference engines accept a single state descriptor as the goal that is needed to be satisfied. Secondly, although it is possible to trace the sequence of rules fired (proven to be true) while attempting to satisfy

the given goal, they do not generate a sequence of actions as the final solution. Thirdly, rule based systems can only add more facts to the knowledge base as new facts are inferred. Goal Regression can delete facts from the knowledge base by making them false as a result of executing an action. The proposed knowledge processing scheme can be visualized with Figure 16. In that Figure, unassigned and assigned orders change state through the application of **Assign** and **Cancel** actions, respectively.

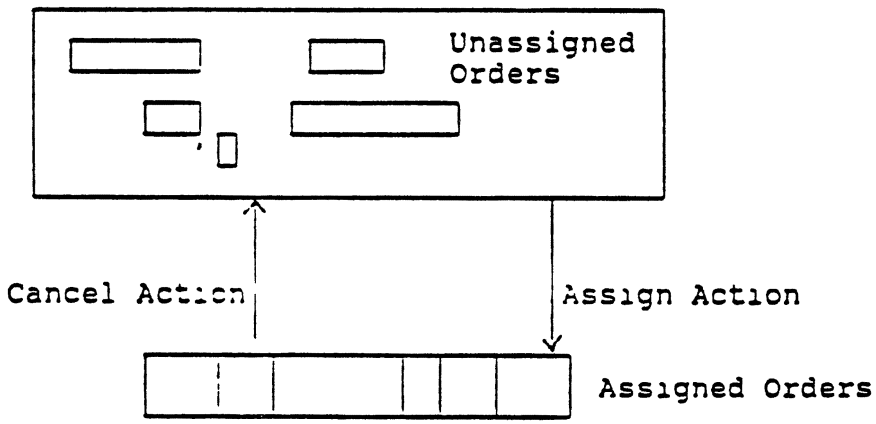


Figure 16, Example Knowledge Processing Scheme

As a complete example of the goal regression method, consider the following manufacturing system example. The initial state is defined as the set of unassigned task orders waiting to be assigned to a machine tool r. The estimated processing times for task orders x, y, z are 3, 5, 2, respectively. Thus, the initial state is represented with the following set:

{Unassigned(x), Unassigned(y), Unassigned(z)}

The goal is defined as finishing each task order at or before the specified due date and is expressed with the set below:

{ Assigned(x,t_x), Assigned(y,t_y), Assigned(z,t_z),
(t_x + estPt(x) ≤ duedate(x)), (t_y + estPt(y) ≤ duedate(y)), (t_z + estPt(z) ≤ duedate(z)) }

The previously defined two actions have the following prerequisites, positive and negative effects:

Pre(Assign(o_i,r,t_s)) = { Available(r,t_s,t_f), Unassigned(o_i), (t_s+estPt(o_i) ≤ duedate(o_i)) }
Add(Assign(o_1,r,t_s)) = { Assigned(o_i,r,t_s), UnAvailable(r,t_s,t_s+estPt(o_i)) }
Del(Assign(o_1,r,t_s)) = { Unassigned(o_i), Available(r,t_s,t_s+estPt(o_i)) }
Pre(Cancel(o_1,r,t_s)) = { Assigned(o_i,r,t_s), (t_s+estPt(o_1) ≤ duedate(o_i)) }
Add(Cancel(o_i,r,t_s)) = { Unassigned(o_i), Available(r,t_s,t_s+estPt(o_i)) }
Del(Cancel(o_1,r,t_s)) = { Assigned(o_i,r,t_s), UnAvailable(r,t_s,t_s+estPt(o_i)) }

Figure 17 presents a graphical view of a portion of the search space for this problem. In Figure 17, the goal state is defined at the top. From the given goal state only three actions are qualified for goal regression because the intersection of the goal set and the negative effects of these actions is an empty set. These actions are assigning orders 1, 2, and 3. In Figure 17, only the branch of the search space that is related to assigning order 1 is presented. The subgoal sets that result from regressing the

goal through the actions are shown below each action that is qualified for regression. The left most subgoal set can be abandoned because it does not meet the due date for order z. Since the assignment actions are given priority over cancellations, assign(z,3) action is explored next and this leads to a solution because the resulting set is the same as the given initial state set. As soon as the initial state is reached, the sequence of assign actions defines the plan to satisfy the given goal state.

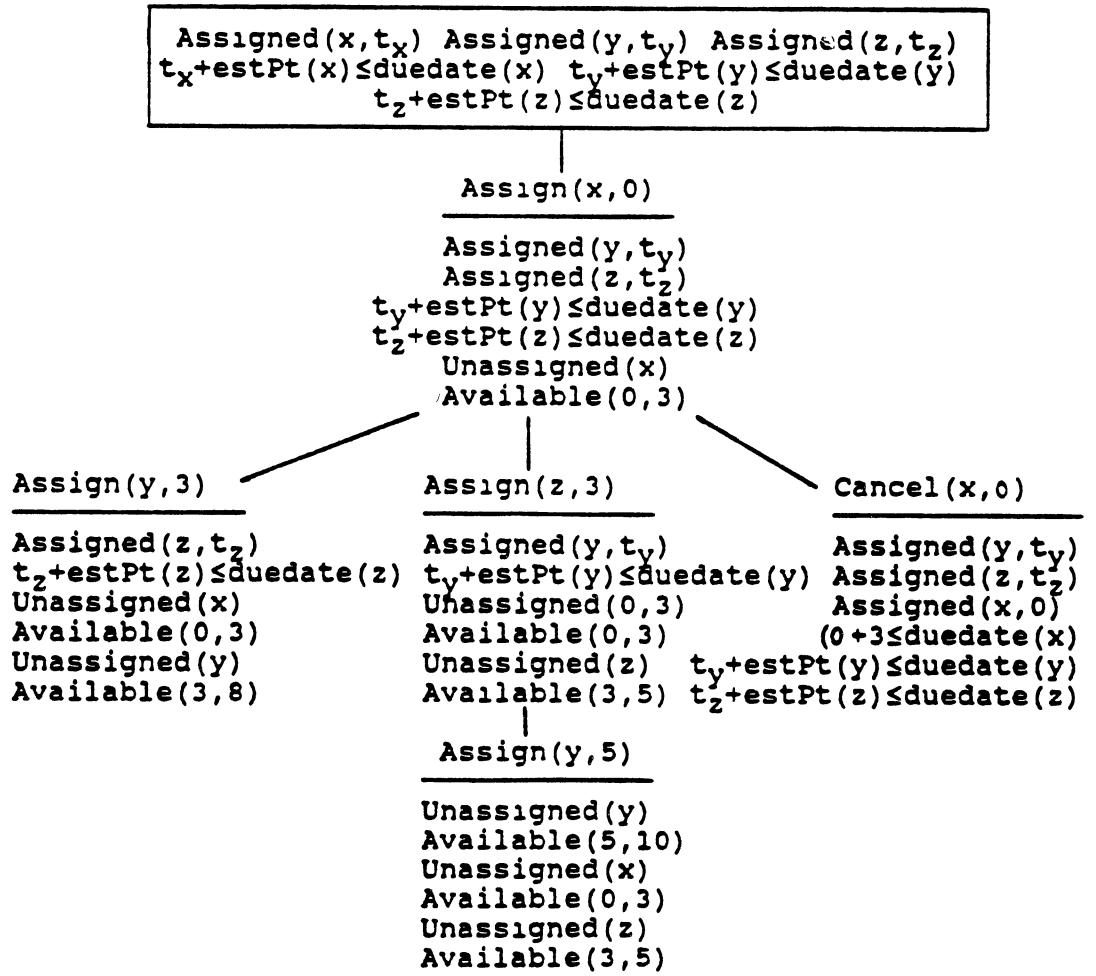


Figure 17, Goal Regression Search Space Example

Besides regular planning activities, the decision making entities engage in exception handling in their **decision** functions. For each type of exception defined for a particular decision making entity, a set of state descriptors and permissible actions are also provided. Each exception event of this type of entity is described in terms of a desired state that will remove the effects of the exception. For example, in the case of breakdowns, the desired state is defined as cancelling all assignments to the affected resource for an estimated period of repair time and assigning those orders, whose due date is before the estimated return time of the resource, to alternative resources. Then, a planning process similar to the one discussed in this section takes place.

In this chapter the operation of **action** and **decision** functions of the formalism are outlined. During the explanation of these functions it is said that the orders, since they are perceived as goals received from the superior level, are the main drivers of the **action** and **decision** functions. The proposed knowledge processing framework for function **decision** (goal regression based) is not yet available in a computer executable form and requires more than the capabilities of present knowledge based software shells. Therefore, only the conflict resolution mechanism, which is the selection of the best action among possible actions, is implemented during the software development part

of this research. The next chapter deals with the issues related to these orders and outlines a formal structure to break orders down into more specific orders that can be understood by lower level decision making entities and finally by information level entities. In a sense, the next chapter outlines the formal communication language between system entities.

CHAPTER VII

GOAL DECOMPOSITION FOR MULTIPLE LEVEL DECISION PROCESSING

Introduction

This chapter addresses the issues related to goals and objectives involved in manufacturing system simulation. The main topic presented in this chapter is the framework for decomposing system goals into subgoals, translating and associating subgoals to decision makers, and finally, the process of linking subgoals to the reasoning scheme defined in the previous chapter. But before proceeding further, the definition of some key terms is in order due to the rather confusing use of these terms in the literature. Since AI literature mainly employs a syntactic definition for these terms, the following definitions, which are more semantically oriented, are adapted from Zeleny (1974).

Attributes are the descriptors of objective reality. They define the characteristics of an object. In the object oriented programming world view, attributes can be expressed in terms of instance and class variables associated with each instance or class entity.

Objectives are closely identifiable with the decision

maker's needs and desires. They represent directions of improvement along individual attributes or complexes of attributes. An attribute becomes an objective when it is assigned a purpose, a direction of desirability or improvement. Attributes are inputs for postulating one's objectives.

Goals are fully identifiable with a decision maker's needs and desires. They are a priori determined, specific values or levels defined in terms of either attributes or objectives. For example, minimizing flowtime for a product is an objective, but achieving a flowtime of 2 days for that product is a goal. It is a specific reference value for that objective. Goals refer to particular target levels of achievement which can be defined in terms of both attributes and objectives.

Criteria are measures, rules and standards that guide decision making. They are all those attributes, objectives, or goals which have been judged relevant in a given decision situation by a particular decision maker.

As can be seen from the above definitions, attributes play a key role in expressing a decision maker's objectives and goals. Furthermore, criteria are determined by a particular decision maker or a group of decision makers and may change from situation to situation according to the decision maker's mode of viewing decision problems.

The following section is introduced to clarify the distinction between the simulation objectives and the system

objectives within the context of this research.

Simulation Objectives and System Objectives

Simulation objectives attempt to answer the question "why are we conducting the simulation study in the first place?". Generally speaking, simulation is used for the following purposes:

1 - Performance comparison of predetermined system structures or behaviors: This type of application is used after the decision maker is confident about alternative system configurations in terms of both structure and behavior. The objective of the simulation study is to assess performance predictions for these alternatives, compare them, and then make a selection among the alternatives considered.

2 - Prediction of absolute performance for a given system structure and behavior: The purpose is to estimate how well a system can perform under a given fixed configuration. This type of study also reveals the limits of the system configuration under study.

3 - Detecting bottlenecks and searching for solutions for a given system structure and behavior: This type of application corresponds to the true problem solving approach. The aim of simulation is to detect shortcomings of the given configuration and to search for the corrective actions that will take the system into a

desirable state in terms of behavior or structure. Contrary to the previous purpose, in this type of study system configuration is not finalized and can be improved under the guidance of system goals and objectives.

The above simulation objectives define the purpose of the simulation study. They set up the experiments for simulation. Different applications require different strategies in terms of the simulation experiments that are going to be conducted. They outline the strategies for how to search for parameter combinations.

In this research, the term system objective is used to express the purpose of the system that is going to be modeled and simulated. As noted earlier, systems that include intelligent entities are purposeful systems, in that they try to achieve some objectives and goals. These goals and objectives are the reasons for the system's existence. A good example of a purposeful system is a manufacturing system. The main purposes of a manufacturing system are defined a priori by system designers and users. Thus, the strategic level objectives can be considered as the main driving thrust of a manufacturing system. The system tries to satisfy these goals or objectives. In the traditional cost oriented manufacturing system view, the main objective is usually defined as maximizing profit while minimizing various costs associated with production. Besides this objective, recent market trends cause strategic decision makers to realize that there are some other objectives such

as maximizing customer satisfaction and minimizing system response time. They are as important as the main objective in order to survive in the market. The problem with these second group of objectives is that they cannot be quantified and measured as easily as cost.

After outlining the difference between simulation objectives and system objectives, the first section of this chapter deals with the problem of how to represent system objectives and goals in accordance with the formalism proposed and how to decompose them into subgoals. Since the studies related to simulation objectives mainly deal with experimental design and automating the simulation life cycle, they are not the main focus of interest in this research. Although it is assumed that the user interface layer will be the main module for the acquisition of the system objectives from the user, it is simulation software's responsibility to represent them in a form that is comprehensible with the proposed formalism and reasoning scheme.

Formal Representation of System Objectives

In order to represent system objectives, a relevant set of attributes and/or criteria must be defined. As stated before, attributes can be expressed in terms of instance and class variables in OOP. The set of attributes and performance criteria (or a single attribute and/or

criterion) that are conceived as relevant to the decision maker's objectives can be defined either in terms of the attributes of the entities controlled by the decision maker or in terms of the decision maker's own attributes. In this research, it is assumed that each decision making entity has its own set of objectives defined before hand. While these objectives are provided by the system designer/modeler, goals that are associated with each decision making entity are the results of decomposing higher level goals. This reveals that the primary interest is in the representation of predetermined objectives and performance criteria along with a goal decomposition scheme. To achieve such a formal representation, some ideas are borrowed from formal language theory for the theoretical structuring of this task. Williams and Upton (1989) propose use of formal language theory for the description of task and control in manufacturing systems. The following section gives a brief introduction to formal language theory along with a summary of their work.

Formal Language Theory

In formal language theory, a grammar G is abstracted with the following four tuple.

$$G = (V, T, P, S)$$

Where,

V is the finite set of variables (non-terminal symbols),

T is the finite set of terminal symbols,

P is a finite set of productions, and

S is the start symbol.

A string is an ordered collection of symbols. The symbols that belong to set V can be replaced by other symbols. This replacement is done according to the rules defined in set P. These rules, or productions, are referred to as re-write rules of the language. The elements of set T represent the lowest level symbols that cannot be transformed into any other symbol. Start symbols are used to define unique starting points for productions. For example, consider the following grammar:

G = (V, T, P, S)

where, V = (S,A,B) and T = (a,b) with productions below.

P = (S --> Ab, A --> a, B -->abb)

A given string SABB is transformed first into

AbABB

with the application of the first production. Then, the second production converts this new string into

abaBB Finally, application of the third production transforms the string into

abaabbabb

Williams and Upton also define some properties such as disaggregation and sequentiality as the necessary requirements on a grammar to be used for manufacturing

systems. Disaggregation refers to breaking higher level manufacturing task symbols into their lower level components, and is analogous to the goal decomposition term of this research. Sequentiality and precedence must be incorporated in order to retain order among symbols. They accommodate sequentiality with the introduction of parentheses. Parentheses are utilized to imply that there is no precedence within the substring. Thus,

(abc)de

denotes that there is no precedence among a, b, and c, but all three must precede d. Furthermore, Williams and Upton's work defines a hierarchical relationship between the symbols of a formal language which is not covered in other similar studies. This property is heavily utilized in this research.

Goal Decomposition

This research perceives the language of a manufacturing system as the abstract representation of the system goals. In the reasoning scheme proposed in chapter VI, system goals correspond to the orders communicated between system entities. A product order, issued by a product level decision making entity, defines the goal for job level decision making entities. Job orders created by job level intelligent entities are actually the goals for task level decision making entities. When task orders are assigned on

individual processing or assembly devices a set of activity orders are triggered. As can be seen, orders from an upper level are perceived as goals for a particular entity and that entity performs its own cognitive activities (reasoning mechanism), searching for a set of orders to the subordinate level to satisfy the orders received. The method defined here takes a high level order and transforms it into lower level orders that can be understandable by the next level decision making entities. In the context of manufacturing systems, non-terminal symbols (or variables) of a grammar can be thought of as higher level goals. These goals can be hierarchically transformed into lower level goals and finally into a set of terminal goals that need to be satisfied by lowest level decision makers. The activities of these lowest level decision makers are oriented toward the satisfaction of terminal goals. Deterministic activity plans are finally triggered by lowest level decision makers and this eventually transforms terminal goals into executable physical actions. For example, let us assume that the top level system objective is defined as

" Meet delivery dates for product orders."

Notice that this is the direction of desire on product orders. It defines the state that the top level decision maker would like to be in. When the product order number and the due date are specified the above objective becomes a goal. This goal is also the starting symbol S for the language defined at this level. Formal statement of the

same goal is

S ===> Complete product order PO100 by XX/XX/XX date

At this point, we need to convert a product order into job orders using the product structure or BOM data. The symbols required for the representation of a product order are defined as product order identification (I), quantity (Q), and due date (D). In AI style of programming these symbols can be thought of as the predicates that define a product order.

The grammar like structure G for the proposed language is defined as

$$G = (V, T, P, S)$$

with, V = (S, I, Q, D, A)

T = (a,b,c,d,e,f,g,h,i,.....,(,), ϵ, τ)

P = (S ---> I

I ---> (A) A | τ

A ---> I

A ---> ϵ

A ---> (X Q

Q ---> Y D

D ---> Z) A

X ---> a | b | c | .....

Y ---> d | e | f | .....

Z ---> g | h | i | .....)

The lower case letters represents the job identification, job quantity, and job due date symbols. The symbols (and)

are used to represent precedence relations, the symbol ϵ is the empty input and symbol τ corresponds to the symbol known as epsilon in formal language theory and represents the terminating symbol. The first production transforms the given goal into product order identification symbol I. The second production transforms symbol I into symbols (, A,), A and τ. In this transformation, variable symbol A is used for the abstraction of a set of job orders. The above production can be explained as grouping of those job orders that can and cannot be processed simultaneously. Transformation to τ defines the end of goal decomposition. The third production is used for conserving overall precedence between job order groups and the fourth one is for indicating that there are no more job orders in this group. The fifth production takes the first job order and starts transforming it into more specific information. The precedence symbol (is used to show that there is no order relation in regard to detailed information symbols of a job order. The unique job order identification is generated through the use of the eighth production, and at the same time it attaches symbol Q for translating the job order quantity. The sixth production works very similar to the third one and transforms product order quantity to job order quantity, but does not specify any parentheses symbols since we are still dealing with the same job order. The seventh production defines how to access the job order due date, indicates the end of terminal symbols for that individual

job order by) symbol and refers to the next job order with recursion.

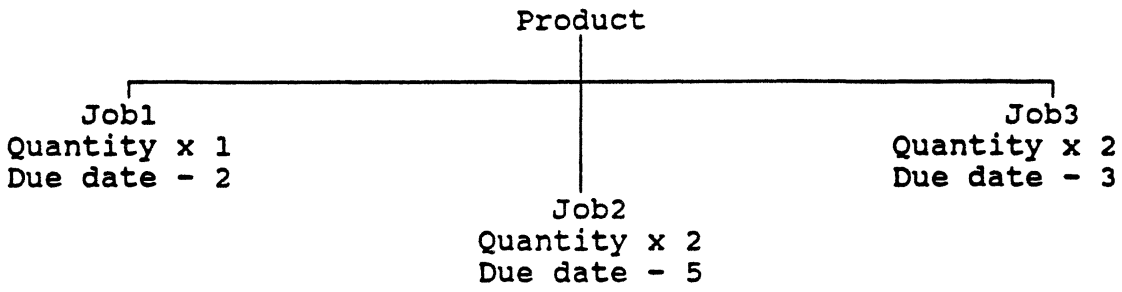
The process outlined here is repeated successively depending on the number of job orders that can be processed simultaneously at the beginning. Once the terminal symbols for those job orders are generated, production two allows us to do the same thing to the other job orders that are following this set. When another set of job orders that has no precedence relation in between is reached, production number three causes the language to repeat the same process. Thus, when the above process is repeated recursively using the product job structure data that also specifies the parallelism among jobs, the original goal is converted into a string of terminal symbols of this level that carries all the necessary information about job orders. Each job order in that string later becomes the starting symbols that correspond to the goals of the next decision level. That string looks like the string presented below:

((a,d,g)(b,e,h)...)(c,f,i)...

The grammar like construct defined in this section is not exactly a formal language grammar because of the way the second and last three productions are defined. For example, in formal language theory, X ---> a | b | c | ..... indicates that variable X can go into a, b, or c, and each time an X is encountered in the input string any of a, b, c, ... is equally qualified for transformation. The structure defined above restricts this aspect of formal language

theory by enforcing a precedence relation in the last three productions. In other words, it says that the first time an X is found that production will take that X into an a, the second time an X is found it will transform it into a b, and so on. In a sense, this concept is the introduction of traditional indexing into the formal language. The second production defined takes the first part of the right hand side for recursion purposes as long as there exist job orders for given product orders. When there are no more job orders, it transforms into an ϵ symbol which specifies the end of the string. Another aspect of the above grammar like construct is that it is context free. Although it is very well known that the manufacturing environment, with its activities and relations, is context sensitive, our purpose here is to develop a generic goal decomposition method. Therefore, the construct defined here allows us to generate hierarchical subgoals from a given goal no matter what the context of the given goal may be. Since most of the previous work in this field utilized formal language concepts for the processing and control of manufacturing systems, they propose a context sensitive language.

The whole concept can be better explained with the following example. Let us define the product structure for a product with the following diagram. In the diagram, it is assumed that jobs that are at the same level can be executed simultaneously.



Now, let us define the grammar G

$$G = (V, T, P, S)$$

with, V = { S, I, Q, D, A }

T = { 1,2,3,a,b,c,α,β,σ,(,) }

P = { S ---> I

I ---> (A) A | τ

A ---> I

A ---> ϵ

A ---> (X Q

Q ---> Y D

D ---> Z) A

X ---> 1 | 3 | 2

Y ---> a | c | b

Z ---> α | σ | β }

where 1, 2, 3 are the job identification symbols for jobs 1, 2, and 3, respectively. a, b, c are the quantities and α, β, σ are the due dates for jobs 1, 2, 3, respectively. Let us define the starting symbol S as the goal of completing 20 units of product M by January 10, 1991. This definition lets us have M, 20, and January 10 values for symbols I, Q, and D, respectively.

The first production of the language defined allows us to access the value of product order identification. Therefore, this production transforms the starting symbol S into the symbol I.

S ---> I

The second production takes symbol I to (A) A. Using the third production, we get the string ((X Q) A. Then, application of the sixth production converts that string into ((1 Q) A. Similar to the process defined above, the following transformations take place sequentially.

((1 Q) A

((1 Y D) A

((1 a D) A

((1 a Z) A) A

((a d α) A) A

((1 a α) (X Q) A

((1 a α) (3 Q) A

((1 a α) (3 Y D) A

((1 a α) (3 c D) A

((1 a α) (3 c Z) A) A

((1 a α) (3 c σ) ϵ) A

((1 a α) (3 c σ) ϵ) (X Q

((1 a α) (3 c σ) ϵ) (2 Q

((1 a α) (3 c σ) ϵ) (2 Y D

((1 a α) (3 c σ) ϵ) (2 b D) A

((1 a α) (3 c σ) ϵ) (2 b Z) A

((1 a α) (3 c σ) ϵ) (2 b β) A

((1 a α) (3 c σ) ϵ) (2 b β) I

((1 a α) (3 c σ) ϵ) (2 b β) τ

As can be seen, the original goal is now decomposed into a string that represents a collection of subgoals. Each (x, y, z) in the above string is a starting symbol for the next level. In other words, we can say that each (x, y, z) is an S symbol for the lower level decision making entity specifying job identification symbol, job quantity, and job due date.

The language like formalism proposed in this chapter (along with the simple illustrative example) is basically a preliminary study to investigate the applicability of formal language concepts for the development of a manufacturing task language. The basic framework proposed in this chapter is utilized during the software development for the translation of orders (goals) communicated between hierarchical control levels.

CHAPTER VIII

IMPLEMENTATION IN OBJECT ORIENTED PROGRAMMING

This chapter addresses the issues related to software implementation of the proposed formalism. It covers the simulation modeling methodology implied by the formalism and OOP implementation. The chapter also includes a brief introduction and discussion of the framework provided with the Smalltalk-80 programming language for event driven simulations.

Introduction

As outlined in section 3.1 of chapter IV, there are several benefits of the OOP approach to simulation. The formalism described for modeling multiple level systems is designed around manufacturing systems. Given the complexity level of today's manufacturing systems, accurate modeling of these systems for simulation purposes can easily become well beyond the capacity of a simulation analyst. This is where the synergistic effect of the formalism and the OOP approach can play a very important role in reducing the conceptual complexity of the modeling task. The formalism outlines the

necessary structures for software implementation which in turn implies a modeling methodology for manufacturing system simulation. Since the detailed OOP implementation of the formalism (chapter V), information and knowledge processing (chapter VI), and goal decomposition (chapter VII) would require several person years of programming effort, a simplified approach is followed for implementation. The simplifications and the assumptions made are explained at appropriate points in the following sections of this chapter. The implementation effort is by no means considered a complete modeling environment. It does serve, however, as a "proof of concept" for the modeling methodologies presented in this dissertation.

Smalltalk-80 Framework for Simulation

The main idea behind the classes defined in Smalltalk-80 (Goldberg and Robson, 1989) for simulation is that the objects that participate in a simulation operate more or less independently of one another. Therefore, it is essential to coordinate or synchronize the activities of the objects involved in the simulation. The simulated objects typically coordinate their activities through the message passing mechanism. However, some objects must coordinate with others at certain times, while some other objects must wait on certain resources that may be unavailable at a given instance before proceeding further in their activities.

Three Smalltalk-80 system classes, namely, Process, Semaphore, and SharedQueue provide the necessary synchronization tools for such situations.

The basic approach followed in Smalltalk-80 simulations is that each of a collection of independent objects with a set of tasks to do coordinates its activity times with those of other objects in the simulated system. Two major classes defined for this purpose are **SimulationObject** and **Simulation**. The class **SimulationObject** describes a general kind of object that might be involved in a simulated system with a set of tasks to do. The main function of an instance of class **Simulation** is to maintain the simulation clock and the queue of events. Furthermore, this class coordinates the arrival of objects to the simulation and resource definitions. The following sections give a very brief description of the main Smalltalk-80 provided simulation classes and their functions. Further information can be obtained from Goldberg and Robson (1989).

class **SimulationObject** :

This class represents any object that can be given a sequence of tasks to do. The instances of this class provide the system defined template for defining real objects involved in the simulation. The subclasses of this class are defined by the modeler and tailored by overwriting the **initialize** and **tasks** methods. As soon as an instance of the modeler defined subclass enters the simulation, the object goes through a general control sequence which

consists of startUp, tasks, and finishUp message protocols. There are several system provided messages, grouped under the protocol named "task language", which any SimulationObject can use in order to describe its tasks. The instances of this class can produce, acquire, release, and inquire about resources defined in the simulation.

class Simulation :

This class represents the engine of the simulation and manages the topology of simulation objects. This class also handles the time advance mechanism of the simulation by scheduling actions to occur according to simulated time. Instances of this class maintain a reference to simulated time, to a queue of events suspended, and to a collection of simulation objects. It is also in this class where arrivals of new simulation objects are scheduled using one of the many scheduling messages. The resources involved in the simulation are also defined here according to their type. The message protocols "modeler's initialization language", and "modeler's task language" provide the standard messages for defining the simulation process. This class utilizes the previously mentioned Semaphore, Process, and SharedQueue classes to implement the time advance and scheduling mechanisms. Creation of new processes in this class gives a sense of independent operation for the activities of each simulation object. The delays and coordinations among simulation objects are implemented by suspending and

resuming these processes through the manipulation of semaphores at appropriate instances of simulation time.

The above two classes are those that are visible to the modeler for defining the simulation system. Another set of classes which are transparent to the user support background activities. The instances of these classes are created, sent messages, and terminated without the modeler noticing them. Simulation modeling with the Smalltalk-80 provided framework is actually programming in Smalltalk rather than a more conventional modeling approach taken in simulation languages like SLAM and GPSS. In this framework, a modeler who is knowledgeable in Smalltalk defines subclasses of the **SimulationObject** class and customizes them by adding more messages or redefining the inherited messages. All of the standard messages defined in **SimulationObject** class are inherited and can be used for defining specific tasks for the instances of this subclass. Next, the modeler defines a subclass of **Simulation** class. This subclass declares how the instances of previously defined **SimulationObject** subclass will enter the simulation and how resources are defined. Two type of resources, called static resources and coordinated resources, can be defined and managed by the subclasses of class **Resource** named **ResourceProvider** and **ResourceCoordinator** respectively. After the subclass for class **Simulation** is completely defined according to the simulation situation on hand, the modeler creates an instance of this subclass and triggers the simulation

process by sending to the subclass the message startUp. This method in turn, triggers the instance creation for the subclasses specified for SimulationObject class and makes them enter the simulation. Then, a cascaded set of messages takes care of the resource creation and their coordination. The simulation objects carry out their tasks under the supervision of the simulation engine and suspend and resume their activities as defined in their tasks. The simulation ends either when the modeler specified ending time or condition is reached or when the event list, which is actually a time sorted list of suspended processes, is empty.

The class hierarchy shown below summarizes the Smalltalk provided simulation classes. The shortcoming of this framework and the modifications made on the hierarchy and some classes will be discussed in detailed in the following sections. The items inside the parentheses give the major instance variables.

Object

Simulation (resources currentTime eventQueue processCount)
  StatisticsWithSimulation (statistics)
SimulationObject ()
  EventMonitor (label)
Resource (pending resourceName)
  ResourceCoordinator (whoIsWaiting)
  ResourceProvider (amountAvailable)
DelayedEvent (resumptionSemaphore resumptionCondition)
  WaitingSimulationObject (amount resource)
SimulationObjectRecord (entranceTime duration)
Histogram (tallyArray lowBound upBound step min max total)

In addition to the above class hierarchy, the Smalltalk-80 simulation framework provides another set of class definitions for random variate generation. The self explanatory hierarchy presented below summarizes the basic statistical distributions provided.

Stream

```
ProbabilityDistribution ()
   ContinuousProbability ()
      Exponential (mu)
         Gamma (N)
      Normal (mu sigma)
      Uniform (startNumber stopNumber)
   DiscreteProbability ()
      Bernoulli (prob)
         Binomial (N)
         Goemetric ()
      Poisson (mu)
      SampleSpace (data)
```

The modifications made in these classes will also be discussed in the following sections. The detailed explanation and code for these classes can be found in Goldberg and Robson (1989).

Formalism and Smalltalk-80 Implementation

This section gives detailed explanation of the relationship between the proposed formalism and the Smalltalk-80 implementation of it. One of the major benefits of formalisms is the clear structural and operational definitions they bring into the software implementation process. The proposed formalism provides unambiguous symbolic representations for entities, concepts and processes that reside in the real system which in turn facilitate the Object Oriented implementation. Using the natural association between these formal structures and OOP, the basic building blocks of the software can be easily defined.

Formal Structures and Software Entities

In the formalism developed, two control levels, namely, data driven physical entities and decision making entities are the components in which all control is assumed to reside in a system. The decisions made at these levels are propagated in the system by going through a translation process at each level and finally being transformed into some form of physical activities. The manufacturing system simulation environment is implemented in Smalltalk-80 using the scheduling and simulation object synchronization mechanisms provided. The major challenge during the

implementation process was to utilize basic tools provided by Smalltalk to define a simulation modeling and processing environment, which is based on the formalism developed, without making extensive modifications in either the formalism or the simulation framework. The following subsections explain the software implementation by making references to the formalism as required to display the role of formalism in defining objects and in the simulation methodology.

The whole simulation environment is designed around two tree structures that are transparent to the user. The first tree represents the hierarchical configuration of the physical entities of the manufacturing system. These physical entities include machine tools, handling devices, etc, and correspond to the **Source System** constructs defined in the formalism. The real physical entities form the leaves of this tree which is named the **SystemStructure** tree. The next level on this tree defines the work center or manufacturing cell configurations. The parent-child relationships of the tree structure defines in which work center each piece of equipment resides and which set of equipment defines a particular work center. The next level on this tree identifies the way work centers are grouped together to form a department or a shop. Each work center in the system points to one shop as its parent and each shop may contain one or more work centers as children. The top level in the tree structure, the root of the tree, defines

the collection of shops defined in the manufacturing system and represents the total system. This type of configuration allows the modeler to define and conceptualize a system in a simple hierarchical form and sets the foundation for an easy graphical model definition interface. Another advantage of this tree structure is the ease of modification in the system structure. As explained later in this section, the separation of physical and control entities of a manufacturing system makes this modification process a relatively easy task. Since the physical and control objects are defined as separate modules, each physical object in the system is in fact a stand alone object unless linked with an appropriate control object.

As soon as each physical entity is defined, the system automatically creates the material channel for this particular physical object which provides for the material flow link between the physical entity and the rest of the physical system. This material channel object corresponds to the material channel, C_{mat}, defined within the Physical system construct, **P**, which is a part of the Source System, **SS**, of the formalism. As defined in the formalism, the main function of the material channel object is to keep track of in and out flow of material for the physical entity with which it is associated. This is also where the material storage, internal to the physical object, is handled. Material channel objects are the facilities that arrange hard linkage among the equipments of the manufacturing

system. All types of interactions with the material take place at material channel objects and these objects provide a default material-equipment interaction mechanism.

In addition to the material channel object associated with each physical entity, the system automatically creates a default control entity for the physical entity and sets up the links between the two. The control entity defines the basic control mechanism connected to this particular physical entity. Each control entity along with its corresponding physical entity, depending in which level it is residing, implements the Information System, **IS**, or the Intelligent System, **INS**, constructs of the formalism. Immediately after the creation of the control object, the system automatically creates a communication channel object for the control object and links them together. The communication channel corresponds to the communication channel structure of the formalism which is denoted by C_{com}. This object provides the soft linkage between the control object and the rest of the system. Since the main input to the control object is information and data rather than material, the communication object is associated with the control object. Using this communication facility, the control object receives orders from upper levels and issues orders to the lower levels. Control objects are the system entities where processing of information and/or knowledge takes place. As stated in previous chapters, the way system representation is visualized in the formalism is that the

control objects, in the form of Information System entities and/or Intelligent System entities, are the key players in a manufacturing system. As a result, almost all of the control activities take place in control objects and these objects are the real active components of the simulation model. The other objects in the model such as physical objects, material channel objects, and communication channel objects support and/or implement the actions of the control objects. The control object classes defined follow the same basic hierarchy given for physical system configuration of the total system and each class has different types of capabilities and functions.

In summary, each node of the tree named **SystemStructure** is built around the physical entity corresponding to that node in the real system, and includes the material channel object and the control object, with its communication channel, that are associated with that specific physical object. Figure 18 shows the **SystemStructure** tree.

In the tree structure given in Figure 18, each node is defined as an instance of AssociationTreeWithParent class and linked to the physical entity defined for the node. The whole tree is implemented as an instance of the class named **SimulationModel**. An instance method of this class is used to construct the tree and this class will be one of the two classes that will interact with the future graphical model definition interface.

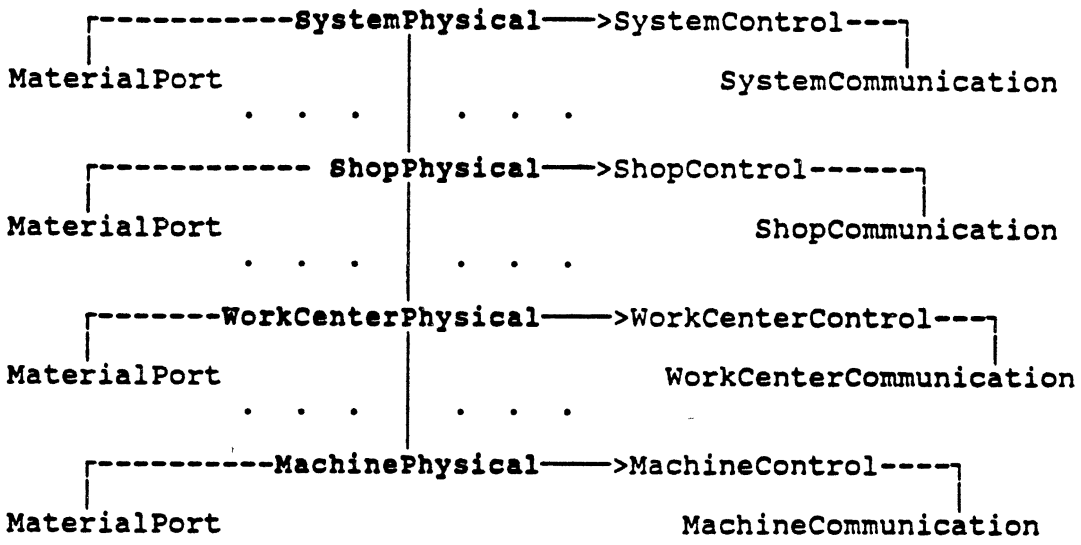


Figure 18, SystemStructure Tree

The second set of information that is essential for the proper representation of a manufacturing system is the product information. A class named **BOMs** is implemented for this purpose. The only instance variable for instances of this class is an association tree node, called products, which denotes the collection of structures for products manufactured in the system. Each child of this node itself is an association tree node which expresses the whole product tree structure for an individual product type. Presently, similar to the system structure definition, this tree needs to be explicitly defined by the modeler and the simulation model automatically creates an instance of **BOMs** class. Along with the class **SimulationModel**, class **BOMs** is the second class that will interact with the future graphical model definition user interface. This tree

construct is referred to as **ProductDataBase**.

SystemStructure tree and products tree are similar to master files which are referred to for accessing and reading important information about system configuration and products.

The **SystemStructure** and products trees explained in previous pages maintain relatively static information about the configuration of the system under study and the products manufactured in it. In order to keep track of the dynamic information involved in the simulation of manufacturing systems, another tree structure is employed. Contrary to **SystemStructure** and **ProductDataBase** trees, this third tree is a highly dynamic one and is frequently manipulated during the simulation. Since it is assumed that the information components are the real driving forces of a manufacturing simulation model, this tree plays a crucial role in simulation processing. In a sense, it represents the dynamic data base of the whole system where transactions processed generate new information, delete unnecessary information, or update present information. This tree, named **OrderDataBase**, holds all the information that is relevant to the orders involved in the operation of the manufacturing system. These orders actually constitute the major portion of the communication between system entities. In addition to the various production orders involved in the manufacturing activities, this tree contains some important information about the customers. Customers represent the

external information flowing into the model in the form of customer orders.

The root of the **OrderDataBase** tree is implemented as an association tree node and corresponds to an instance variable of SystemControl object. This style of implementation allows the system level control object to easily access most of the dynamic information about the customer and production orders. The children nodes defined right below the root contain a unique customer identification in the form of a customer label (name). Each of these customer names is associated with an instance of a class named **IDGenerator** which stamps each customer order with an unique identification number according to its order of arrival. This association object, customer name and IDGnerator pair, itself is implemented as a tree node and each children node of it represents the individual order of the customer specified with the customer name. These individual orders are also defined as tree nodes and consist of an identification number and a time value pair. The time in the association represent the time when this particular order is entered into the system. The next level in the tree represents the product combinations for each individual customer order. Each product type at this level is also a tree node which points to a unique "identification number-entrance time" node. Furthermore, each node at this level is associated with an instance of class Dictionary. This

dictionary acts as a small local data base and contains information very specific to the individual customer order and product type.

An important aspect of the **OrderDataBase** tree is that at the beginning of the simulation this is basically an empty tree consisting of a root linked to SystemControl object. As the customers arrive to the manufacturing system, the branches and subbranches of the tree representing unique customer names, their individual orders, and specific product combinations requested within each individual order are dynamically created as the simulation progresses. The system level control object is the place where translation from customer orders to specific product orders takes place and this tree is manipulated at those levels mentioned up to now. The subsequent levels of the tree are also dynamically created using the information stored in **ProductDataBase** by appropriate level control entities. As soon as a product order is created and added to the **OrderDataBase** tree, the system control entity decides on releasing or not releasing this order to the appropriate shop. If the order is released, shop level control object goes through another translation process and creates the required component orders for the product and adds them to the appropriate nodes of the **OrderDataBase** tree. Thus, the system level control object acts as a gate to control the flow of product orders, derived from customer orders, to the shop control entities. Similarly, by deciding on releasing

or not releasing component orders, shop level control entities act like a component order flow gate which controls the arrival of component orders to work center control entities. When a component order is released from a shop control entity, the work center control entity receiving the order goes through a translation process to create the necessary part orders and adds them to the node in the **OrderDataBase** that represents the component order received. Finally, the part orders, called BatchOrders, created by work center control entities are released to the lowest level of control, machine control entities. This interaction corresponds to the activity orders described in Chapter VI. Each machine level control entity communicates with a physical machine entity and sends to it the necessary signals (messages) to perform the manufacturing operation. This is where the real connection from the logical system to the physical system takes place.

As a result of the process described above, as the simulation clock advances, the OrderDataBase tree grows. The status of individual product, component, and part orders are updated by the control entities as operations are performed on them. When the final assembly operation of a product order is completed, the branch of the tree originating from that node is chopped off and related statistics are updated. This process prevents infinite growth of the OrderDataBase tree. The deletion process from the tree is implemented this way because of the localized

variable information kept in individual component and part orders. Thus, at any instance in time before the completion of the final assembly of a product order, orderDataBase can be querried about the history and the status of each component and part order associated with this particular product order. Figure 19 graphically represents the structure of the OrderDataBase tree. In Figure 19, the dictionary objects associated with each order object contain the local information about the order object and the root node is the "customers" instance variable of the system level control object.

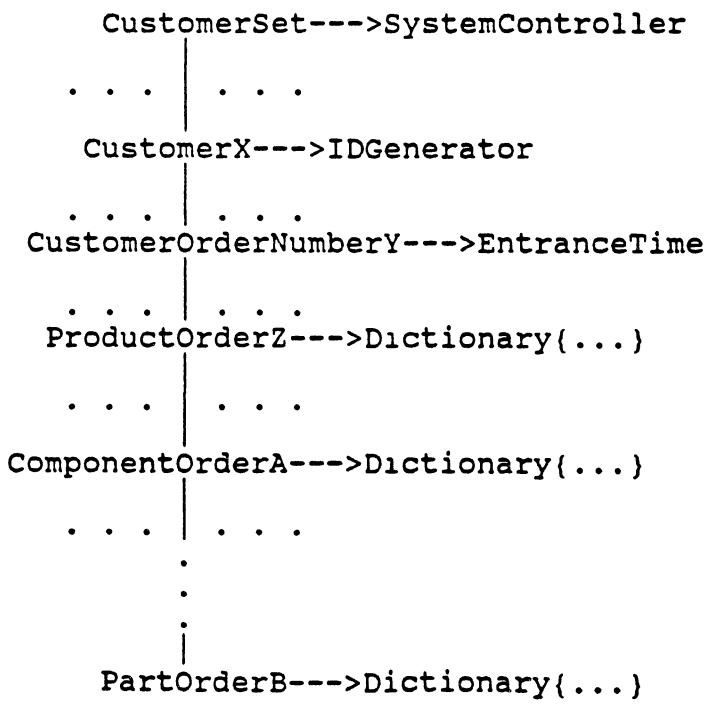


Figure 19, OrderDataBase Tree

Once all the component and part orders of a product order are released by the control objects involved in the production of the product, the simulation proceeds as follow; As the operations are performed on the parts or on the components by machine physical objects, associated machine controllers are informed about the progress of the operation orders. If the operation is performed on a part, which is the first transformation step from the raw material, the work center controller involved is informed. The work center controller then checks the status of all part orders of the parent component order, and if all are completed informs the supervising shop controller. This shop controller in turn, decides on proceeding further or not. If the operation performed on the physical machine is a component operation, a message flows directly from machine controller to work center controller and from work center controller to shop controller. The shop controller determines what to do with the information. If all the components of the product are completed, the shop controller passes this information to the system level controller. When system controller is informed that the final assembly operation of the product is finished, it updates the product order statistics and manipulates the **OrderDataBase** tree to reflect this event.

The process described above can best be visualized in terms of asynchronous waves of messages between the control objects of the system structure hierarchical tree. The

whole chain is triggered by the arrival of a customer. The system level control object translates the customer order into a set of order specific product orders and decides whether to release these product orders. When a product order is released, the shop responsible for handling these type orders receives the order and translates it into a set of specific component orders for the product and makes decisions about the release of these component orders by synchronizing them. A similar process, but with a different content, takes place at work center and machine control objects, and finally the message wave reaches the bottom of the hierarchy, to the physical machine objects. This process forms the downward message wave. As operations on parts and components are performed at physical machine objects, an upward message wave takes place. This set of messages originates from the physical machine object and may go up to work center, shop, or system controller level through the hierarchy depending on the operation performed and the present status of the parts and components.

Figure 20 summarizes the flow of orders and hierarchical structure of the communication taking place during the simulation.

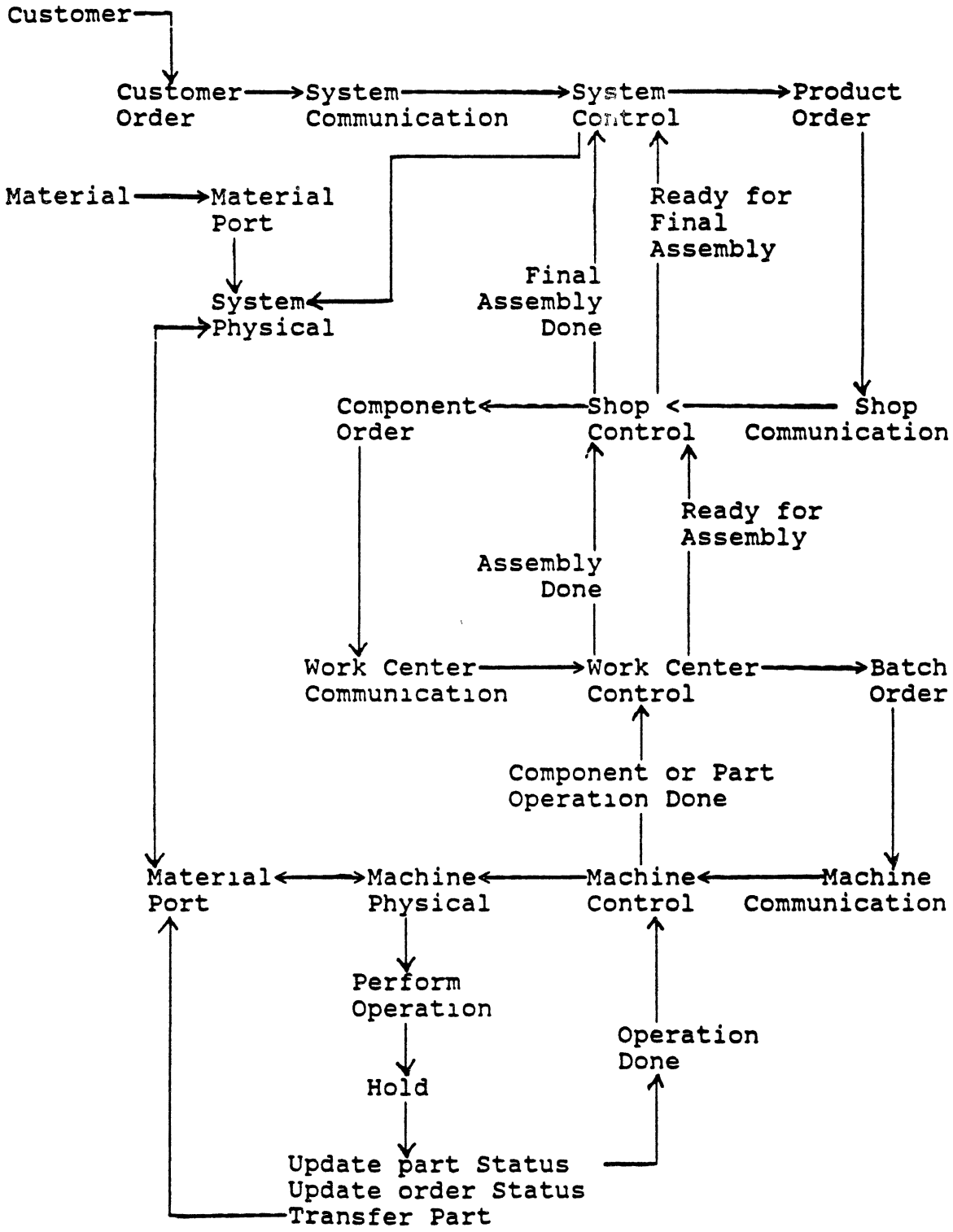


Figure 20, Order Flow and Communication

Smalltalk Classes Defined

This section gives a brief description of the classes defined in Smalltalk-80. Detailed implementation information on these classes along with the program listings can be obtained in Appendix A. These classes implement the formalism presented in chapter V, the information and knowledge processing presented in chapter VI in a simplified form, and the goal decomposition method proposed in chapter VII in a modified form. The Smalltalk simulation classes cast the foundation for the classes defined here. Most of the classes defined use the system provided scheduling and object coordination mechanism. The simulation control classes are defined as a subclass to the class **Simulation** and customized according to the framework defined in chapters V, VI, VII. Similarly, most of the physical and logical entities involved in manufacturing system simulation are defined as subclasses of the class **SimulationObject** to implement the formalism. In summary, these classes define another layer specific to the framework developed that sits on top of the Smalltalk provided simulation framework. The following tree structure summarizes the hierarchy of the classes defined.

```
Object

   Simulation
      Simul

   SimulationObject
      SystemPhys
         ShopPhys
            WorkCenterPhys
               MachinePhys
      SystemCntrl
         ShopCntrl
            WorkCenterCntrl
                MachineCntrl
      SystemCommunication
         ShopCom
            WCCom
               MachCom
      MaterialPort
      ProductOrder
         CmptOrder
            BtchOrder
      RawMaterial
      Cstmr

   BOMs
   SimulationModel
   IDGenerator
   QueueStatistics
   UtilizationStatistics


class SimulationModel :

     instance variables - none

     message protocol - Model definition
                           defineModel
                        Accessing
                           systemStructure
```

The instances of this class contain information about the way the physical system is configured. The only method of Model Definition protocol is presently edited by the modeler to define the structure of the particular physical system being modeled. The code in this method basically creates the nodes that correspond to each physical entity

involved in the system. When a graphical user interface is developed for model definition, this method will be automatically edited by the user interface. The other instance method of this class, systemStructure, returns the root of the systemStructure tree. This provides an easy entry point for accessing system configuration information.

class BOMs :

instance variables - products

message protocol - Initialization
initialize
defineProducts
Accessing
products
product:node:

The only instance variable of this class, products, forms the root of the tree that contains product structure subtrees for each product type manufactured in the facility. This root is initialized through the initialize method defined. The other method of the Initialization protocol, defineProducts, is the method where individual product trees are created and connected to the root. In the present state of the software implementation this method is also edited by the user. When an interactive graphical interface is developed, this method will be automatically edited by the user interface. The products method of Accessing protocol provides an entry from the root of the tree for accessing information contained in the tree. The other access method allows entry from the root of a particular product structure tree.

Class **Simul** :

instance variables - none

message protocol - Initialization
defineArrivalSchedule
defineResources

This class is defined as a subclass to the class **Simulation** and its instance defines the arrival schedule of two types of external entities to the system namely, raw materials and customers through **defineArrivalSchedule** method. Furthermore, it defines the various types of product, component, and batch orders as coordinated resources in defineResources method. All order objects in the system are implemented as coordinated resources to provide for the synchronization between the release and capture of an order. The instances of this class inherit **resources**, **currentTime**, **eventQueue**, and **processCount** instance variables to implement the simulation processing mechanism for the manufacturing system simulation framework.

class **SystemPhys** :

instance variables - controller, node, matChannel

message protocol - Initialization
initialize
controllerEnterSimulation
Accessing
controller
getNode
setNode:
materialChannel
checkForMaterial:
Task Language
commitMaterial:quantity:
distribute:quantities:to:

The instance variables used in this class define the controller as the instance of this class, a reference to the node in the **SystemStructure** tree which holds this physical system object, and a pointer to the materialPort object defined for physical system object. The initialize method is where the controller (SystemCntrl) and materialPort for system level physical object is created and associated with this physical object. The method named controllerEnterSimulation is used to define the general mechanism by which various levels of controller objects, system, shop, workCenter, and machine enter the simulation and is inherited by the subclasses of this class.

Access protocol holds various type of methods to access or define the various objects linked with the physical system object. Furthermore, checkForMaterial method of this protocol queries the material port about the availability of a particular type of material in system level material storage. As mentioned previously, the system level physical entity manipulates the only physical entity flowing into a manufacturing system, material. The commitMaterial:quantity: method makes a certain quantity of a particular material type unavailable for further demand and dedicates that amount to a particular order, and updates the amount available. The other method of the task language makes the initial distribution of the necessary raw material to appropriate physical machines as soon as a product order is released. This method is also inherited

and used by the subclasses of this class for material transfer between physical machine objects.

class **ShopPhys** :

instance variables - none

message protocol - Initialization
initialize

This class inherits all of its instance variable definitions and methods from its super class **SystemPhys** with the exception of initialization method. Initialization method in this class creates the controller and material objects for the physical shop objects.

class **WorkCenterPhys** :

instance variables - none

message protocol - Initialization
initialize

WorkCenterPhys class definition is very similar to its super class **ShopPhys**, the only difference being the context of initialization method where material port and controller are created for physical work center objects.

class **MachinePhys** :

instance variables - load, utilization, procsTimeDist, timeToBreak, mttbDist, state
message protocol - Initialization
initialize
Accessing
load
resetLoad
status
utilization
Task Language

```
breakDown
doOperation:
getBatchTime:
loadMaterial:quantity:
loadMaterials:quantities:
operationDone:
performOperationOn:
unloadMaterial:
tasks
```

This is an important class definition in the implementation. The instances of **MachinePhys** class are the actual implementors of the manufacturing operations on raw material, parts, and components. Contrary to the system level physical entity, which represents the totality of the physical entities in the system and the material interactions at the system level, the instances of this class implement material state transformations and interactions with material at the individual machine level. The instance variables hold information about the present material object being worked on, present state, and utilization statistics. In addition, mean time to break distribution, process time distribution, and remaining time to breakdown information are also kept in instance variables. The access protocol defines methods for retrieving the information contained in some of the instance variables.

The task language implements the physical operations on material. The machine control object issues an activity order, described in chapter VI, by sending physical machine object the message "performOperationOn:". This message triggers the ordered set of activities required to satisfy

the request. First, the physical machine object starts with an idle status and loads the material or a set of material depending on the nature of the operation (assembly or regular) by interacting with its material port. Next, it calculates the time required to process the whole batch using batch quantity and operation time per part information by interacting with the products data base. Once the batch time is calculated for the operation requested, it compares this batch time against time to break. If batch time is less than time to break, after updating time to break it issues a "doOperation:" command to itself. The method "doOperation:" is the place where physical machine state change, statistical updates, and time delay take place. If batch time is greater than time to break, it starts the operation till breakdown time and schedules a breakdown signal to itself when time to break reaches zero. When a breakdown is encountered, physical machine object immediately reports to its controller, changes its state to broken, waits until it is repaired, sets its timeToBreak instance variable to a new value by using its mean time to break distribution, and completes the remaining part of the operation. The next two steps of the ordered set of activities are unloading the processed material and signaling the delivery of it to its next destination, and reporting to controller object about the completion of the operation request.

As can be seen from the above explanation, this class implements the processing of activity orders described in chapter V.

class **MaterialPort** :

```
instance variables - incomingMaterial,
                     outgoingMaterial,
                     matCommited, inQueue, owner
message protocol - Initialization
                      initialize
                      setOwner:
                   Accessing
                      incomingMaterial
                      inQueue
                      outgoingMaterial
                   Task Language
                      tasks
```

The instances of class **MaterialPort** provide the basic interaction tool between the material flowing through the system and the physical entities of the system. They are analogous to communication channels associated with control objects, but they handle material objects instead of information. The instance variables defined for this class are used for storing incoming and outgoing material objects for the physical entity, material committed for orders, queue statistics collection, and a pointer to the physical object. The initialization methods are for initializing the instance variables and setting up the link between the material port instance and the physical object. Access protocol is self explanatory and provides access to instance variables. The task method of the instance continuously monitors the arrival of material by taking advantage of the semaphore based non-busy wait mechanism. As soon as

material arrival is detected, it updates the incoming queue by creating new entries for material types arriving the first time.

```
class SystemCntrl :

     instance variables - controls, status, database, plan,
                  customers, channel, productBeingConsidered
     message protocol - Initialization
                           initialize
                        Task Language
                           askCustomerForOrder:
                           customerArrival:
                           getAction:
                           productDone:
                           readyForAssembly:
                           releaseAssemblyOrder:
                           releaseOrder:
                           translate:for:
                        Accessing
                           controls
                           customers
                           database
                           getDestinationFor:
                           getPlannedOrders
                           getReleasedOrders
                        Reset KB Interrogator
                           moreOf:
                           oneOf:
                           request:
                        Knowledge Acquisition
                           customer
                           product
                           quantity
                           shopCapacityStatus
                           shopLoadStatus
                           slack
                           slackMinusLeadTime
                        Planning and Decision
                           planning:
                           reviewPlan
                        Action Implementation
                           putHimInPlan
                           release
```

This class implements one of the major players of a manufacturing system simulation, the system level control mechanism. All the interaction with the main information

entities that are external to the system, customers, takes place at an instance of this class. The communication channel object associated with an instance of this class provides in and out flow of information to and from the system. The instance variables of the class keep information about the physical system object which an instance of this class is linked to, present status of the instance, its local database and plan, a link to its communication channel object, an order presently considered for decision, and a pointer named "customers" which holds the root of the **OrderDataBase** tree.

The only method in protocol "Initialization" creates an instance of **SystemCommunication** class, establishes the link between two objects, and sets the initial values for instance variables. Most of the methods defined in protocol "Accessing" are self explanatory and provide for retrieving the objects stored in some instance variables. Since most of the methods defined in the task language are inherited by the subclasses of this class, they are generic.

The "customerArrival:" method is triggered by the communication channel (where customer object enters the system) of system controller object and checks the customer unique key against **OrderDataBase**. If it is the first time that this type of customer has arrived, it creates a branch in the tree for the unique customer key. In addition, this method initializes the statistics regarding the arrival of this customer and gets the order from the customer object

through "askCustomerForOrder:" method. This second method retrieves the order from the customer object, creates a unique identification number for the customer order and adds a new node for this type customer in the **OrderDataBase**, and starts the order translation process. The order translation process is actually decomposition of the customer order, which may contain several product orders, into individual product orders. During translation, specific information about each product order such as quantity, due date, lead time, order status, machine name where the final operation is performed, etc. are created using the information from **ProductDataBase** and customer order. This order specific information is stored in an instance of class Dictionary and associated with the product name. This product name and dictionary pair make up the new node added for the parent customer order. After the addition of these new nodes, each representing an individual product order, the planning mechanism is activated to make decisions about this new set of product orders. The planning scheme initiates the expert system based intelligent activities by using "getAction:" method. This implementation style completely conforms to the dynamic aspect of the formalism in terms of making decisions one at a time using present internal and perceived external states. The internal structure and the operation of the expert systems defined for system, shop, and work center level control objects will be discussed in detail in the following sections. The method "getAction:" immediately

sets up the link between the simulation object, system control object, and the Humble knowledge base defined for system level control by changing knowledge base interrogator to the system control object. The protocol "Reset KB Interrogator" provides the necessary methods for this purpose. Once the link is established, the system control object waits for the action proposed by the expert system. The methods in protocol "Knowledge Acquisition" supply simulation real time knowledge to the expert system in terms of present status of the system. The expert system activates these methods as the backward chaining mechanism tries to find values for its system state definition variables.

As soon as a decision is made about what to do with each product order, the methods of "Action Implementation" protocol implement the action. This corresponds to update internal and update external functions of the formalism and reflects the results of the action taken. Presently, only two type of actions regarding orders are implemented. These actions are referred to as release and putHimInPlan and are self explanatory. If an order is put in the plan where orders are scheduled according to their due dates, it will be re-evaluated when another decision problem is encountered. If the decision is to release the product order being considered, the system controller changes the status of the order, creates a productOrder object, defines its content, requests its physical counterpart to distribute

the committed material to appropriate physical machine objects, and enters a new productOrder object into the simulation. After the product order is released, system controller object goes through its plan and re-evaluates each product order against changed system state and time.

The other methods defined for the task language of this class are used to process the progress reports received from shop level control entities. The message "readyForAssembly:" is sent by a shop control entity signaling that all the components of an assembly are done and waiting. In light of this new information, shop control entity creates another instance of productOrder with different status and content.

```
class ShopCntrl :

     instance variables - newComponents, loadStatus,
                          capacityStatus
     message protocol - Initialization
                           initialize
                        Accessing
                           capacityStatus
                           loadStatus
                           getDestinationFor:
                        Task Language
                           assemblyDone:
                           getAction:
                           orderArrival:
                           readyForAssembly:
                           releaseOrder:
                           tasks
                           translate:
                           translateMore:with:
                        Knowledge Acquisition
                           component
                           componentStatus
                           quantity
                           slackMinusLeadTime
                           workCenterCapacityStatus
                           workCenterLoadStatus
```

This class, which is a subclass of **SystemCntrl** class, reimplements some of the methods defined in its super class within the context of shop control objects. The instance variables keep track of the information about the load and capacity status of an individual shop. After instance variables are initialized, the instances of this class interact with their supervising system control entity through their shop communication objects. These shop level communication objects provide the coordination of product orders between system control object and shop control objects. When a shop control object is signalled by its communication channel object about the arrival of a product order, it immediately looks at the order content and checks the status of the order. If it is an order about a final assembly operation, the shop control object communicates this new order to its proper work center controller by creating and releasing a component order with this new information as order content. If it a newly released product order, the shop control entity triggers its own translation process. The translation process at this level corresponds to exploration of the ProductStructure data base for the components and creating the proper nodes in the **OrderDataBase** that are equipped with special information derived from the newly received product order. The method "translateMore:" recursively operates to explore the components of a product structure tree. The next step of the shop control object is the planning process to decide on

releasing or not releasing the corresponding component orders for this newly generated node of the **OrderDataBase** tree. The mechanics of the planning process here are very similar to the system level control entity, the major difference being the new type of knowledge acquired and the different knowledge base used for the expert system. The simulation real time knowledge used at this level is basically about the load and capacity status of the work centers controlled by the shop and the important attributes of the component being considered. The Shop Planning expert system, which is covered in the following sections, imitates the typical reasoning mechanism used in shop control problems and executes within the context of each shop. If the decision is to release an order to start working on the component being considered, shop control object creates an instance of class **CmptOrder** and releases it, which is captured by the communication channels of subordinate work center controllers. If the decision is to not release the component order, the component order is put into the shop controller's plan ordered according to the component order's due date. As assembly operations are performed on component orders, work center controllers report these events to the supervising shop controllers where further information is generated and may be reported to the system controller, if components are ready for final assembly operation. The general operation of the shop control object is similar to the operation of the system control object.

class WorkCenterCntrl :

instance variables - none

message protocol - Initialization
initialize
Accessing
capacityStatus
loadStatus
Task Language
componentDone:
getAction:
orderArrival:
partDone:
releaseOrder:
tasks
translate:
Knowledge Acquisition
machineCapacityStatus
machineLoadStatus
part
partStatus
slackMinusLeadTime

This class implements the last intelligent control level in the hierarchy defined by the formalism. Its function and operation is similar to its super class **ShopCntrl**, but with a completely different context. The instances of this class receive component orders from their superior shop control objects and decide on what to do with these orders. If a component order is released for the first time, the work center controller translates it into a set of part orders, updates **OrderDataBase** tree and goes through its planning process. The planning process is similar to the one employed by upper control objects. Thus, most of the planning methods are inherited from super classes. Although planning processes are similar, this class consults with its own expert system defined for the work center problem domain. Based on the action proposed by

the expert system, work center control object implements the action through the methods inherited from the above classes. If the decision is to release the translated part order to the associated machine controller, work center controller creates an instance of **BtchOrder**, batch order, sets its content and releases it. Otherwise, translated part orders are put in a plan to be acted upon at a later time.

class **MachineCntrl** :

instance variables - none

message protocol - Initialization
initialize
Accessing
status
Task Language
orderArrival:
reportBreakdown
tasks
translate:

The instances of this class provide the link from upper level intelligent control entities to data driven information entities of the formalism. The main function of machine control objects is to receive batch orders from their superior work center control objects, check for material by querying the material port through physical machine object, and issuing a request to its physical counterpart to trigger the set of order actions for performing the requested operation.

class **SystemCommunication** :

instance variables - inMessages, owner

message protocol - Initialization
initialize

```
Accessing
    inMessages
    owner
    setOwner:
    getTime
    setTime:
Task Language
    tasks
```

An instance of this class is used as the coordination mechanism between the customer objects and the system control object. Instance variable inMessages is used to keep track of customer arrival history and owner is to set up a link between system communication object and system control object. The main task of this object is to continuously monitor customer arrivals to the system using the semaphore based object coordination. As soon as a customer arrives, this object acquires the customer and signals its owner, transfers the customer to him, and returns to monitoring.

```
class ShopCom :

    instance variables - orderTypeSought

    message protocol - Accessing
                           lookFor
                           orderTypeSought
                       Task Language
                           startSeekingOrder
```

An instance of this class is associated with a shop control object. Contrary to system communication class where the only entities monitored were customers, this class provides the setup to seek different types of product orders. The instance variable orderTypeSought is fixed by the shop control object during the creation of the instance

and monitoring is triggered by the controller. The continuous monitoring mechanism defined here is similar to its super class, but out of all product orders released, each instance monitors a particular type. As soon as an instance of product order type sought is released, the shopCom instance seizes and transfers it to the associated shop control object.

A subclass of class **ShopCom** is defined and named **WCCom** to provide the coordination between various types of product dependent component orders released by shop control objects and the work center control objects. The instances of this class operate exactly like the instances of **ShopCom** class. Another class defined under **WCCom** class is named **MachCom** and coordinates the release of various types of batch orders by work center controllers and seizure of these orders by machine control objects. Again, the operation and function of the instances of this later class is similar to its super class, the only difference being the order objects sought now are various types of batch orders.

class **PrdctOrder** :

instance variables - content

message protocol - Accessing
getContent
setContent:
Task Language
tasks

The instances of this product order class along with its subclasses **CmptOrder**, and **BtchOrder** implement component

and batch orders, respectively. Instances of these classes represent the information flow that takes place in the system and forms the communication bond between various levels of control objects. The definition of these tree classes are very similar to each other, the only difference being reimplementation of the task method which defines what type of order the instance is. The only instance variable, content, is used as the information pocket where control objects define specific order content, which is implemented as a dictionary, and deposit it into this pocket before releasing the order. Once the order is received by a subordinate control object, the content is retrieved and used for translation and knowledge acquisition purposes.

```
class Customer :

     instance variables - name, customerOrder, entryTime

     message protocol - Initialization
                           initialize
                        Accessing
                           entryTime
                           getOrder
                           name
                        Task Language
                           tasks
     class variables - DueDateDistribution,
                       NameDistribution, NumberOfProdDist,
                       ProdDist, QuantDist
     class message protocol - Accessing
                                 DueDateDist
                                 NameDist
                                 NumbOfProdDist
                                 ProdDist
                                 QuantDist
                                 setDueDateDist:
                                 setNameDist:
                                 setNumbOfProdDist:
                                 setProdDist:
                                 setQuantDist:
                                 setDistributions
```

The instances of this class represent the customers arriving to the manufacturing system. It is assumed that the information external to the system flows into the system through customers. The instance variable name defines the unique identification key for the customer. The instance variable customerOrder is analogous to order content and holds information on what types of products are desired, their due dates and quantities. The third instance variable, entryTime is used as a time stamp. The initialize instance method sets up the values for customer name, customer order, and entryTime getting randomly selected values from the empirical probability distributions provided by the modeler. The class variables and methods of customer class are defined for distributions to find how many different products are desired, their due dates, product names and quantities, and customer name.

class **RawMaterial** :

instance variables - type, quantity, entryTime

message protocol - Initialization
initialize
Accessing
entryTime
quantity
type
setQuantity:
setType:

This class provides the material objects flowing through the physical system. Three instance variables contain information on what type of raw material is received, what quantity, and when it is received. The

subclasses defined by the modeler for this class describe the unique material identification names and empirical distributions for quantities. As soon as instances of these subclasses are entered into the simulation, they coordinate with the material port object of the system physical object.

class **IDGenerator** :

instance variables - label, count

```
message protocol - Initialization
                      initializeWith:
                   Accessing
                      generateID
                      reset
class message protocol - Instance Creation
                            newFor:
```

This is a simple class used to assign unique identification numbers for various orders of an individual customer. The instances of this class point to the object that created them and increment their counters by one each time they are referrenced for an identification number.

class **UtilizationStatistics** :

```
instance variables - title, lastState, busySum,
                maxIdle, maxBusy, lastTime, batchCount,
                busySumSqrt
message protocol - Initialization
                      initialize
                   Accessing
                      setTitle:
                      update
                      update:
                   Printing
                      header:
                      titlePrint:
                      printStatisticsOn:
```

This class implements the typical utilization statistics collection process of discrete event simulation and generates the standard report at the end of the simulation. The typical instance variables for this purpose are initialized at the beginning of the simulation and updated through an update operation. The output report is generated by the print protocol upon the modeler's request after the simulation is completed.

class Queuestatistics :

```
instance variables - title, lastTime, cumWait, count,
                     maxLength, cumLengthTime,
                     cumlengthTimeSqr, currentLength
message protocol - Initialization
                      initialize
                      setTitle:
                   Accessing
                      enter:length:
                      exit:length:
                   Printing
                      header:
                      titlePrint:
                      printStatisticsOn:
```

The instance variables of this class keep track of important variables to generate a queue statistics report upon the modeler's request. The material objects register with an instance of this class when they enter and exit a queue. The print protocol defines the methods to generate a standard queue statistics report.

The following segment of Smalltalk code is used to invoke the whole simulation process:

```
|model aSimulation stat machStat
machStatFile queStatFile statFile|

(1)  Cstmr setDistributions.
(2)  aSimulation _ Simul new startUp.
(3)  ProductDataBase _ BOMs new.
(4)  ProductDataBase initialize defineProducts.
(5)  model _ SimulationModel new.
(6)  model defineSystem.
(7)  [aSimulation time <43200] whileTrue: [aSimulation proceed].
(8)  statFile _ (FileStream fileNamed:
                        'c:\st80\image\results\Order.sts').
(9)  stat _ SystemStructure value controller database.
(10) stat associationsDo: [:each | each key = #ProductsStatistics
                        ifTrue: [each value setTitle2: '']
                        ifFalse: [each value setTitle2: (each key)].
                        each value printStatisticsOn: statFile].
(11) machStat _ SystemStructure leaves.
(12) machStatFile _ (FileStream fileNamed:
                              'c:\st80\image\results\Machines.sts').
(13) machStat do: [:each | each value utilization
                               printStatisticsOn: machStatFile].
(14) queStatFile _ (FileStream fileNamed:
                             'c:\st80\image\results\Queues.sts').
(15) machStat do: [:each | each value materialChannel
                               inQueue setTitle: (each key).
                             each value materialChannel inQueue
                               printStatisticsOn: queStatFile].
```

A brief explanation of each line is as follow:

(1) This line is used to set up random number seeds of the several distributions defined for class **Cstmr**.

(2) An instance of class **Simul** is created and sent the message **startUp**. This instance is the key object of simulation where simulation processing takes place.

(3) & (4) A new instance of class **BOMs** is created and set to a global variable for easy access. The method **defineProducts** creates the user defined product structure trees for the products manufactured in the system.

(5) & (6) An instance of class **SimulationModel** is created and

method defineModel generates the modeler defined **SystemStructure** tree which is used to express the configuration of the physical entities of the system.

(7) This is where the simulation clock is checked against user specified termination time. As long as the simulation clock is less than the termination time, the simulation proceeds.

(8) - (15) These lines define the file names and paths for printing the standard reports on utilization, queue, and order statistics.

Modifications Made to Smalltalk and Simulation Framework

Some modifications are made to the Smalltalk-80 programming environment and to its simulation related classes. The detailed information on methods added and/or reimplemented can be seen in Appendix B. The changes made are grouped under five broad classes.

1 - The scheduling mechanism of the standard simulation framework provided by the Smalltalk system is a semaphore and process based mechanism. Although there exists an event queue, it is not the same as traditional event calendars used in discrete event simulation processing. This queue contains time ordered delayed event instances that are uniquely identified with their semaphores. These delayed event instances actually represent a list of interrupted processes which are later resumed by sending signal messages to their

associated semaphores. Since it does not allow a direct link between the interrupted process and the simulation object with which it is associated, this style of implementation makes preemption of simulation objects impossible. A preemption mechanism is added to the framework by defining a new subclass as follows:

class PreemptableSimulationObject :

super class - SimulationObject

instance variable - delayedEvent

message protocol - Task Language
holdForWithPreempt:pass:
preempt
setDelayEvent:

The following methods are also added to class Simulation:

delayFor:for:
delayUntil:for:

The instances of this class reimplement some methods of their super class SimulationObject, thereby allowing preemptions. The only instance variable is used as a link between the simulation object and the delayed event object. When an instance of this class schedules a time delay by its "holdForWithPreempt:pass:" method, a pointer is set between this object and the delayed event object through a set of cascaded messages. When this object is asked to preempt its activity via the preempt method, event queue is accessed (through a newly defined method) from the presently active instance of class Simul and searched for the instance variable delayed event. The condition, which is actually the time to resume, of this delayed event is set to the present

time. Thus, the delayed event corresponding to the activities of the preemptable simulation object becomes the first delayed event to be resumed. More detail on this class can be found in Appendix B.1.

2 - The second group of changes involve the operation of Smalltalk provided probability distributions. The way these probability distributions are designed and used in simulation does not allow control over seed number specification. The class variable U, which is an instance of class **Random**, defined for class **ProbabilityDistribution** is the mechanism where random number streams are generated for probability distributions. The main problem with this implementation style is that a single random number generator which uses the values obtained from the computer's clock is used for all probability distribution subclasses. This makes the random aspect of a typical simulation uncontrollable and makes duplication of experimental conditions impossible. This problem is solved by defining a new instance variable called "generator" in class **ProbabilityDistribution** and reimplementing method "next" in each probability distribution subclass. Thus, instead of inheriting the random variable generation method "next" from class **ProbabilityDistribution**, each instance of a probability distribution class uses its own generator to obtain a value with uniform probability distribution over the interval [0,1]. This allows specification of a different random number seed for each

instance of a probability distribution class, making generation of identical random variable streams between experiments possible (Also see Appendix B.2).

3 - The third group of changes is made by adding more versatility to the Association tree framework. A recursive method is added to the class **AssociationTree** to explore and return all leaves of any given tree. This property is used by control objects during their manipulation of **OrderDataBase** tree. The details about this modification can be seen in Appendix B.3.

4 - The Histogram class of Smalltalk is changed to add the standard deviation capability to the histograms generated. In addition to number of objects, minimum, maximum, and average values, instances of histogram class calculates and prints the standard deviation of the values observed (Also see Appendix B.4).

5 - The last set of changes to Smalltalk are contained in Appendix B.5. The classes **Resource** and **ResourceCoordinator** are changed to control the processing sequence of batch orders by machine control object when the work center knowledge base is employed. Certain methods are added/changed in **ResourceCoordinator** to support control over the Smalltalk provided resource queue named pending.

Intelligent Manufacturing System Entities and Knowledge Bases

The Non-programmed knowledge processing, in the form of planning through goal regression, proposed in the knowledge processing section of chapter VI, requires a high degree of sophistication and is beyond the capacity of the Smalltalk-80 expert system shell provided. Consequently, a very simple form of non-programmed control scheme is implemented as a demonstration of the general idea. First of all, goal regression outlines how knowledge processing in the form of planning can be handled by intelligent entities. The mechanism itself is theoretical and rather complex to implement in software and is also beyond the limits of this research activity. Secondly, it requires conclusions to be drawn on a set of parameters rather than a single parameter, a capability which is not available on any expert system shell, Smalltalk-80 based or otherwise. The non-programmed decision making scheme implemented for the intelligent control entities of the proposed framework is based on the idea that each time an order is received from upper level, the control object decides on what to do with it. The possible actions that can be taken are problem domain dependent and relative to the expert system defined for that problem domain. Two basic actions implemented for the expert systems designed on an example system modeled are explained in the following chapter.

The following pages give a brief explanation of the relationships between the Humble expert system framework and the intelligent entities of the developed simulation software. Humble is a Smalltalk-80 based expert system shell developed by Xerox special information systems (Humble reference Manual, 1987) and is used as the implementation tool for the non-programmed decision making aspect of intelligent entities. As may be recalled, in the software developed, intelligent entities of a manufacturing system correspond to the system, shop, and work center level control objects. The machine level control basically handles preprogrammed decision situations which are directly implemented in Smalltalk-80 programming language.

In Humble's object oriented style, the development of an expert system begins with the identification of the Entities, or objects, about which the system will make inferences. Entities fall into categories called entity types and are characterized by Parameters. Knowledge base parameters defined over these entities describe how entities of the same type differ from one another. In a Humble knowledge base, in addition to parameters, an entity can hold other entities. These entities, called sub-entities, have their own set of parameters and sub-entities and so on. The parameters of an entity are where the real information about that entity lies. Every parameter in a knowledge base can contain data which is uncertain. This means that each parameter may have some attached value which indicates how strongly the system

believes that the value is the correct one for that parameter. These certainty factors are numbers ranging from -1.0 to 1.0. The system takes these factors into account whenever it draws a conclusion or makes a test. A 1.0 indicates absolute certainty that the value is the correct one. -1.0 indicates that it is absolutely certain that the value is not the correct one for whatever parameter it is associated with. 0.0 indicates a complete lack of knowledge. The system supports the ability to have several different potential values for each parameter in the fact base simultaneously. Each of these potential values is referred to as an hypothesis. The best hypothesis is the hypothesis with the greatest certainty attached to it at that time. Included in the static knowledge are descriptions of what sorts of entities can exist in the fact base, as well as how to use the information in the fact base to make inferences and thus create new information. This description of what sorts of inferences are permissible is called the rule base. Each rule describes under what conditions a given conclusion can be drawn. The collection of entity types along with their parameters define the framework in which some part of the world can be described, and therefore have rules written about it.

The design of non-programmed decision making for an intelligent entity begins with the design of a Humble knowledge base corresponding to that entity's intelligent behavior. As outlined above, it is the process of defining

entities that will be involved in the inference process, their parameters, and a set of rules that will define how to draw conclusions. Before linking the designed knowledge base to the simulation, a knowledge engineer (software designer; in the future this activity will be a part of model definition via a friendly user interface) can test the behavior of the knowledge base as a stand alone expert by querying it. Once it is decided that the knowledge base is behaving as intended, it can be readily used in simulation. The knowledge engineer can always change the knowledge base by editing it (redefining rules, adding new entities, parameters or rules, deleting existing entities, parameters and rules), and this last version of the knowledge base will be used by the simulation.

The interaction between the intelligent simulation object and the knowledge base takes place as follow: When a controller object needs to make a non-programmed decision it triggers the method called **getAction** from its task language protocol. The hierarchical control entities (system, shop, and work center controls) overwrite this method, thereby allowing the use of a different knowledge base at each level. The method **getAction** first finds the specified knowledge base among all the knowledge bases defined. Next, it initializes the knowledge base entities, in other words, resets the knowledge base's fact base by initializing the values of the parameters. This is necessary because each knowledge base retains its last facts status, and during the simulation each

time a knowledge base is consulted by a control object, knowledge (facts) is dynamically acquired from the executing simulation model and must not contain values from a previous consultation. Furthermore, presently at each control level, a generic knowledge base that applies to all entities of a level is defined allowing the same knowledge base to be used by different control objects at the same level. Therefore, each consulting object needs to dynamically acquire his own domain specific knowledge during the interaction.

The next message sent to the knowledge base from the control object causes initialization of knowledge base entity types. This message resets the counters used to make up default knowledge base entity names. The next thing the control object needs to do is to set itself as the interrogator of the knowledge base and set the knowledge base's output stream to nil. This means that the simulation object will query the knowledge base and conclusions drawn will be sent back to the simulation object.

Finally, the control object triggers the interrogation process by sending **findOut:** message to the knowledge base with parameter **action** being the argument. The knowledge base tries to find out a value (a proposed action) to that parameter by using its backward chaining mechanism. During the backward chaining mechanism as the value of a parameter in the knowledge base (such as shop status, etc.) is required in one of the fired rules, the knowledge base automatically refers back to the interrogating control object to acquire

this piece of information. The control object in turn uses the appropriate method defined in its knowledge acquisition protocol to retrieve on line data, process that data, create the knowledge, and return it to the knowledge base. During the data retrieval process, the control object may interact with the subordinate level entities to extract the necessary data. Depending on how much knowledge needs to be acquired, this process is repeated for every piece of on-line knowledge acquired and may require heavy interaction between the knowledge base and the interrogating control entity. Since Humble has the uncertainty property, there may be more than one value qualified as the possible action for parameter action. Among all qualified values, the one associated with the highest certainty value is returned by the knowledge base back to the control object as the action needed to be taken by the control object.

When the concluded action is passed back, the control object implements the returned action using the corresponding method defined in its action implementation protocol. The whole process described above is repeated each time a knowledge base is interrogated by one of the control objects.

CHAPTER IX

EVALUATION OF THE APPROACH THROUGH AN EXAMPLE MODEL AND ANALYTIC HIERARCHY PROCESS

This chapter summarizes the characteristics of an example system modeled and the statistical analysis of the results obtained from the model. In addition, the Analytic Hierarchy Process is used to compare the proposed modeling and simulation approach against conventional simulation paradigms.

An Example Manufacturing System

At this phase of the research, for demonstration purposes, a fictional manufacturing system with produce-to-order operational policy is defined and modeled using the concepts developed in chapter V and the software explained in chapter VIII. This model also shows the modeling methodology that is indirectly dictated by the formalism and the software. The example model demonstrates the versatility of the information and insight that can be gained from the model along with some idea on how closely the model can mimic the real system.

In the developed framework, modeling begins by describing the physical configuration of the manufacturing

system. This physical system configuration forms the backbone of the simulation and is the process of defining machine tools that are contained at each work center, work centers that are included in each shop, and shops or departments that form the total manufacturing system. The present state of the model definition facility requires programming in Smalltalk-80, but will evolve into a graphical user interface facility in future extensions. Model definition takes place in an instance of class **SimulationModel** described in chapter VIII and basically is the process of creating **SystemStructure** tree nodes with unique identification keys defined for each machine, work center, and shop. Figure 21 graphically depicts the physical configuration of the example manufacturing system being used in this research.

As described in chapter VIII, as soon as these physical entities are created their corresponding default material ports and control objects (with their communication channels) are automatically created. When these objects are created and linked to the nodes, the system structure tree grows and takes a form similar to the one presented in Figure 14 of chapter VIII. Each of these default entities, depending on their type, contain default processes to handle material, information, and control procedures. The software modules that will allow tailoring of these default entities is left as a future extension plan. The future graphical user interface tools will provide the transparent objects

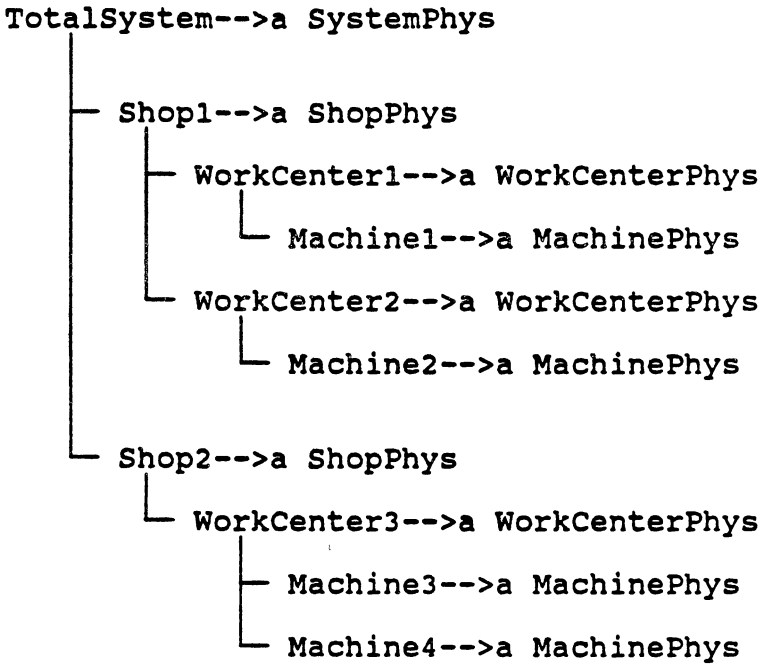


Figure 21, Example System's Physical Configuration

and classes to customize these default entities as their physical counterparts are defined by the model developer.

After the system structure is defined for the example system, the next step is the definition of some example products that are manufactured in the system in terms of their product structures and product data bases. Since it is assumed that the system produces on order, for simplification purpose only two types of products are defined. Their product structures are pictorially presented in Figures 22 and 23.

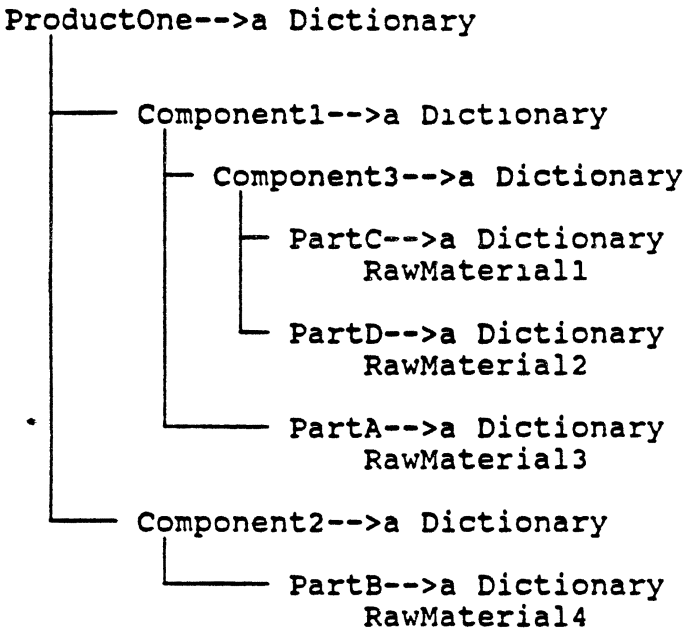


Figure 22, Example Product Structure

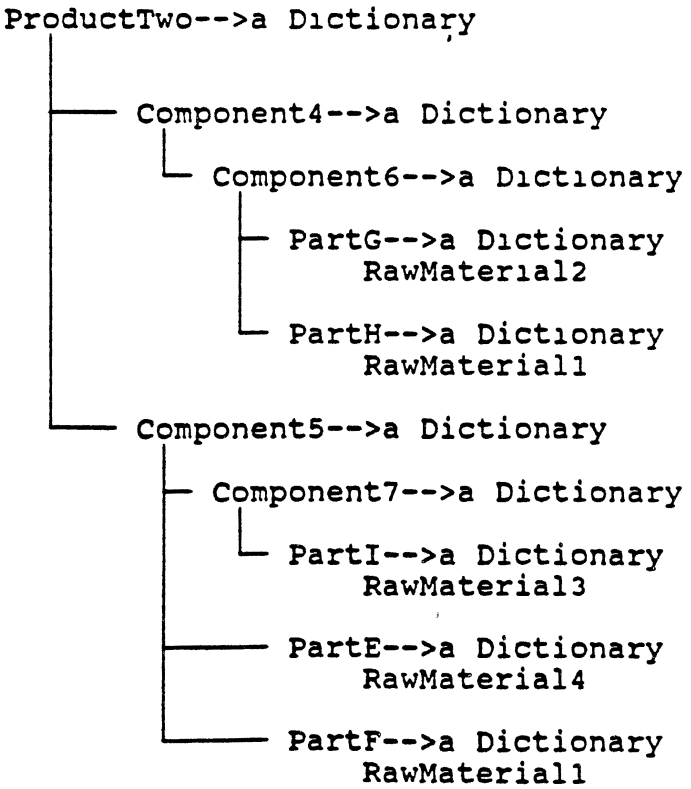


Figure 23, Example Product Structure

In those Figures, the dictionaries are the objects where all information regarding that particular product, component, or part is kept for reference. These product structure trees with their information dictionaries linked to a root node called "products" (which is an instance variable in systemControl object) form the total product data base. Although there are slight differences from one object type to another, data dictionaries associated with each node of the product data base typically contain the following information:

- Machine name, where the operation (processing or assembly) that produces this part, component, or product is performed
- Mean operation time for the operation
- Lead time for the part, component, or product
- A factor (number) which indicates how many of this entity is required for each of its parents in the product structure tree.

The tree structure described above is defined using an instance of **BOMs** class. Similar to **SimulationModel** class, the definition of this class will be transparently handled by the graphical user interface in the future.

The next step in modeling activities is the definition of the generic expert system for each control level. Three Humble expert systems called **systemPlaning**, **shopPlaning**, and **WorkCenterPlaning** are defined to be used by system, shop, and work center control objects, respectively. Since these

expert systems need to be system specific, in real modeling situations the simulation model developer must either have enough expertise on operational policies or rules of each level in the manufacturing system or need to interact with the real experts of each level. This aspect is important for realistic representation of the non-programmed decision making processes involved in the system. The example expert systems defined for the manufacturing system outlined in previous pages are rather simple ones and reflect typical behavior of hierarchical control levels of a make-to-order type of operation.

System level expert (Appendix C.1) has two knowledge base entities named planner and shop with various parameters affiliated with each. Upon receiving the request from system controller, **systemPlaning** knowledge base first checks the timing of the product order. If timing is critically low, it immediately releases the order. Otherwise, it goes through a set of rules to derive the importance of the order. The knowledge used in this derivation process is the order size, product type and the unique customer identification key. Although, no discrimination is made among customers and products in the present version of the expert system, all the structure for that purpose is defined and can be easily adjusted to prioritize customers and/or products. For the sake of simplicity (at the cost of losing this kind of advantage of expert system), it is assumed that all customers and products are equally important. In

reality, depending on profit margins and customer records, orders received from the customers can be graded using the set of rules provided in the knowledge base (with minor changes). Once the order is uncertainly rated w.r.t. one of the three classes, namely, "notImportant", "important", and "veryImportant", the next step is the determination of the order timing. The timing is defined as a function of order importance, lead time, and slack time. Depending on which timing zone (each timing zone is defined in terms of certain multiples of order lead time) the order falls into, it is given timing of "urgent", "normal", or "notUrgent". As soon as the timing is determined, the expert system acquires the shop load and capacity knowledge from the interrogating object and searches through a set of rules to find a value for the parameter called **action**. The data regarding the shop load and capacity (mostly analytical) is retrieved by the interrogating system control entity, processed by this entity, converted into a symbolic form, and passed back to the expert system during the interrogation. The conversion of analytical data to symbolic knowledge takes place as follow:

The system controller looks at the shop load in terms of a moving monthly time window (bucket). As product orders are released, the windows of the shops involved are loaded with proper time values derived from lead time information and the order quantity. As product orders are completed, the time windows of involved shops are again updated

(decremented) with proper time values. When shop load status is requested, the load of that particular shop at that instant of time is retrieved and divided by the fixed time window length. Depending on into which one of the three defined regions this ratio falls, one of the possible symbolic values is returned to the expert system. Three possible values defined for shop load are; "Heavy", "Medium", and "NotHeavy".

A symbolic value for the shop capacity status is determined using the simple ratio of number of machines operational in the shop at that instant of time divided by total number of machines of the shop. Again, three different regions are defined for this ratio, each associated with one of the 'High', 'Medium', or 'Low' symbolic values.

The next set of rules ascertain the possible values for the parameter "action" according to various combinations of order timing, shop load status, and shop capacity status situations. In each case, the value assigned to "action" (either release or putHimInPlan) has an uncertainty value associated with it.

The second knowledge base, **shopPlaning** (Appendix C.2), is defined for non-programmed shop decisions. The two knowledge base entities defined are shopPlanner and workCenter, each with a set of parameters. This expert system deals with component orders and its main design principles are similar to those in **systemPlaning** expert. It

assesses a value for the parameter called timing utilizing the information obtained on component due date and lead time from interrogating shop control object. Again, three possible values (urgent, gettingClose, canWait) are defined for timing using some multiples of component lead time. A set of rules classify the component on hand according to its position in the product structure tree. The knowledge regarding the particular work center that is responsible for the production of this component is obtained through parameters called workCenterLoadStatus and workCenterCapacityStatus. The method used in deriving the symbolic values for these parameters is very similar to **shopPlaning** expert, with the main difference being the weekly time window used in assessing the work load of a workCenter.

This expert system searches through a set of rules representing various combinations of workCenterLoadStatus, workCenterCapacityStatus, and timing parameters and finds a value for "action" with a certainty factor. This value is passed back to shop controller which in turn implements that action.

The last non-programmed decision level in the hierarchy is handled by the **workCenterPlaning** knowledge base which deals with batch orders released to machine controllers (Appendix C.3). This is a relatively simple expert system that consists of two knowledge base entities WorkCenter and Machine, each with a set of parameters. The rules defined

here first check the status of the machine on which the operation is going to be performed. If the machine is broken at that instance, it holds on to the batch order until the machine is repaired. If the machine is operational, then it derives a value for time status of the batch order using various multiples of the batch processing time and batch due date. If timing is below a critical limit, it immediately returns "release" as the answer to the query. Otherwise, depending on which time range the batch falls into, it sets one of the "immediate", "attention", or "alert" values as batch timing and finds a value for "action" using this batch timing.

The workCenter control object in turn implements this action either by putting the batch order in the plan or releasing it to the corresponding machine controller. Additionally, this level dictates the machine controller to consider parts and components that are ready for assembly before the others.

The other system relevant data is as follows;

Customer Order Data

- Customer names are generated using a sample space with A, B, and C being the possible customer names.
- Number of products requested is a sample space with 1 and 2 values.
- Product names is also a sample space with ProductOne and ProductTwo as the possible values.

- Order sizes are generated using a sample space of (20, 25, 30, 35, 40).
- Due dates for each product ordered is from a sample space of (1500, 2500, 3500, 5500, 7500) minutes.
- Customer inter arrival times has an exponential distribution with 1100 minutes mean time.

Physical Machine Data

- Processing times are normally distributed. Mean of the normal distribution is obtained from the product data base each time a processing operation is requested. Standard deviation is fixed and assumed as 1 minute. More detail on individual process times can be obtained from the BOMs class defineProducts method where the data dictionaries for each part, component, and product is defined (also see Appendix A). Individual processing times are mostly between 5 and 10 minutes per item.
- Mean time to breakdown is defined as an exponential distribution with a mean of 20,000 minutes.
- Mean time to repair is also an exponential with 1500 minute mean time.

The simulation model of the manufacturing system with its three knowledge bases described in the previous section is simulated for a three month period (43,200 minutes = 480 minutes/day). A one factor four levels experiment is designed to compare the performance of the system with or without different levels of knowledge based control.

Considering the make-to-order nature of the manufacturing system, the following three statistics are defined and collected as the main performance measures:

1) Manufacturing Velocity : This is the time elapsed between the release of a product order by the system level controller and the completion of that order. Smaller values of manufacturing velocity are desirable since this is the measurement of how fast the system can satisfy a given order.
2) Order Lateness : Time difference between the completion of a product order and its due date. Negative values correspond to early completion. This is also an important performance measurement for make-to-order type of operational policy.
3) Customer Response Time : Time between the arrival of a customer to the system and the completion of all products requested by the customer. Again, smaller values on this measure are more preferable and show how fast the system can respond to the customer's total demand.

In addition to these statistics, traditional statistics of manufacturing system simulation, such as machine utilizations and queue statistics are collected. These statistics are not used as the main system measures but kept as a reference. Among the three measures defined above, manufacturing velocity is selected as the main performance measure to be analyzed, since it better relates to

traditional performance measures. The four experimental levels are defined as follow:

Level 1

Manufacturing system with no non-programmed control. In this control scheme, the orders at all levels are translated to suborders and immediately released to the next level. The orders mostly accumulate at the machine control level where they are processed according to FIFO.

Level 2

Only system level non-programmed control is employed. The release of product orders to the next level is controlled by the system controller using **SystemPlaning** knowledge base. Each time a customer arrives or a product is completed, this expert system evaluates all the orders not yet released (including the new arrivals) starting with the earliest due date order. The lower level controls are not changed and FIFO is still kept for the lower levels of control.

Level 3

The system and shop control objects both use their knowledge bases simultaneously to employ non-programmed decision making at their control levels. Similar to system controller, shop controller interrogates its knowledge base each time a component is completed or a product order is released from system control level. The component orders not yet released are also considered, starting with the earliest due date.

Level 4

The system, shop, and work center control objects all employ their knowledge bases simultaneously. Similar to system and shop controllers, work center consults its knowledge base each time a batch order is completed or a component order is released from shop controller. Again, batch orders are evaluated according to their due dates. Additionally, this control scheme tells machine controller to consider product and component batch assembly orders before other processing operations.

The main idea behind these example knowledge bases is to perform a tighter timing control around product, component, and batch orders as they are moved downward in the hierarchy. It can also be noticed that the frequency of consultations with the relevant knowledge bases is different for each control level. The work center knowledge base is the most frequently queried one, shop knowledge base second, and system knowledge base is the least.

Table I shows the results (in minutes) of five replications made for each level (also see Appendix D). Common random number seeds are used to induce a negative covariance, thereby reducing the variance during pair wise comparison of the difference between manufacturing velocities of levels. The random number seeds are changed between replications but kept the same across the levels. Furthermore, to better observe the effect of non-programmed control, the manufacturing system is slightly overloaded.

In other words, in the long run the orders will tend to accumulate.

The simulation data is analyzed as follow:
First Level 1 is compared with Level 2 in terms of manufacturing velocity by taking the difference between mean manufacturing velocity values for runs 1-2, 5-6, 9-10, 13-14, and 17-18.

Level 1	Level 2	D_1, Level 1 - Level 2
11,825	11,335	490
14,017	13,645	372
15,377	14,747	630
14,861	14,259	602
14,425	11,182	243
	Average	467.40
	Std.	161.65

Then, a 95% confidence interval is constructed for the true mean difference. ($t_{4,.025}=2.78$)

$$467.40 \pm 2.78\ (161.65/\sqrt{5}) = (\ 266.43,\ 668.37\ )$$

The above confidence interval for true mean difference does not contain zero, and completely lies in the positive values region. Therefore, we can conclude that the difference is statistically significant. This means that, the manufacturing system has a smaller value for manufacturing velocity when system level knowledge based control is employed. In other words, the system can produce slightly faster with the expert system. The reason for the difference being small is because of the diminishing effect of system level control to the shop floor as orders are moved down in the manufacturing system's control hierarchy.

TABLE I

SUMMARY OF SIMULATION RUNS

Levels	No Expert	System Expert	System & Shop Exp.	System & Shop & Work Ct. Expert
Run #	1	2	3	4
Manuf. Velocity	11,825	11,335	10,488	9,024
Order Lateness	7,058	6,735	6,117	5,045
Cust. Respose T.	14,144	13,696	12,572	10,447
Run #	5	6	7	8
Manuf. Velocity	14,017	13,645	13,788	11,712
Order Lateness	9,209	8,999	9,502	7,440
Cust. Respose T.	16,210	16,265	16,386	13,679
Run #	9	10	11	12
Manuf. Velocity	15,377	14,747	14,415	8,212
Order Lateness	10,478	10,227	9,856	4,262
Cust. Respose T.	18,180	17,876	17,495	8.983
Run #	13	14	15	16
Manuf. Velocity	14,861	14,259	14,743	7,509
Order Lateness	9,980	9,867	10,623	4,123
Cust. Respose T.	16,725	16,552	17,356	8,598
Run #	17	18	19	20
Manuf. Velocity	11,425	11,182	10,965	9,623
Order Lateness	6,544	6,430	6,574	5,375
Cust. Respose T.	13,736	13,610	13,544	11,542

Since it is concluded that system expert performs better than no expert, the second phase of the analysis is the pair wise comparison of level 2 and level 3.

Level 2	Level 3	D_1, Level 2 - Level 3
11,335	10,488	847
13,645	13,788	-143
14,747	14,415	332
14,259	14,743	-484
11,182	10,965	217
	Average	153.80
	Std.	502.78

A 95% confidence interval for the true mean difference is;

$$153.80 \pm 2.78\ (502.78/\sqrt{5}) = (-471.28,\ 778.88)$$

Because this confidence interval contains zero, we can conclude that there is not enough evidence for claiming one level is better than the another. The length of the interval suggests that more replications are needed to reach a conclusion on the true difference. One thing to bear in mind is that the performance of systems with knowledge bases heavily depends on the particular knowledge bases defined. The knowledge base designed for shop level controllers is a generic one and does not take into account the individual differences of the shops. All the knowledge bases designed in this research are for demonstration purposes and restructuring of the shop control knowledge base may yield better results.

Therefore, with the sample on hand we can conclude neither that the system with system controller knowledge

base nor the system with system and shop controller knowledge bases is better than the other.

Because of the conclusion summarized above, both level 2 and level 3 are pair wise compared to level 4.

Level 2	Level 4	D_1, Level 2 - Level 4
11,335	9,024	2,311
13,645	11,712	1,933
14,747	8,212	6,535
14,259	7,509	6,750
11,182	9,623	1,559
	Average	3,817
	Std.	2,593

A 95% confidence interval for the true mean difference is;

$$3{,}817 \pm 2.78\ (2{,}593/\sqrt{5}) = (\ 593.25,\ 7{,}040.75\ )$$

Although relatively wide, this interval does not contain zero and clearly lies in the positive values region. So, with the sample on hand we can claim that manufacturing velocity is significantly faster (with a large variance) when system, shop, and work center levels use their knowledge bases simultaneously.

Level 3	Level 4	D_1, Level 3 - Level 4
10,488	9,024	1,464
13,788	11,712	2,076
14,415	8,212	6,203
14,743	7,509	7,234
11,965	9,623	1,342
	Average	3,663
	Std.	2,826

A 95% confidence interval for the true mean difference is;

$$3{,}663 \pm 2.78\ (2{,}826/\sqrt{5}) = (\ 149.56,\ 7{,}176.44\ )$$

Again, this relatively wide confidence interval completely lies on the right hand side of zero. This observation implies that the system's manufacturing velocity is faster with system, shop, and work center knowledge bases used together as opposed to only system and shop knowledge base use.

These last two conclusions were somewhat expected and suggest that unless a sound non-programmed and programmed control scheme is applied at the shop floor level, the benefits that are going to be obtained from upper level non-programmed control schemes will be marginal and will not significantly impact the overall performance of the system. This fact observed through this new modeling framework closely coincides with most real life situations (Vollmann et al., 1984).

The other major statistics summarized in Table 1, order lateness and customer response time, show a trend similar to manufacturing velocity between levels. It can be observed from the Table data that addition of system knowledge base slightly improves order lateness, but once again, the main improvement comes from the addition of work center knowledge base. In general, manufacturing velocity which is analogous to the more conventional throughput measure can be usually improved by shortest processing time (SPT) rule. But since SPT does not take due date information into account, it may

easily deteriorate order lateness statistics. In other words, they may be conflicting objectives. If we try to improve one, we may get undesired results on the other one. This is where the advantage of knowledge based control comes into play. By using both due date and lead time information along with present state of the system in the knowledge base, we can define rules that make measured compromises. The customer response time statistic is closely related to manufacturing velocity, and shows a parallel trend. As can be observed from Table 1, customer response time is also significantly reduced by the addition of work center knowledge base to the hierarchy of non-programmed control.

The more conventional performance measures such as utilization statistics and queue statistics can be viewed in Appendix D. One observation on this data is that as layers of non-programmed hierarchical control are added, queue lengths "generally" tend to be slightly shorter and machine utilizations "tend" to give somewhat smaller values (because of better batch timing), with some fluctuations with shop control knowledge base. Because shorter queue lengths are more desirable to reduce in process inventories, the above observation is an indication of improvement. On the other hand, traditionally higher utilization values are sought for quick return on investment at the expense of lost flexibility provided by slightly under utilized machines. Thus, in the light of these two conflicting views on equipment utilization, with the sample in hand, we can claim

that non-programmed control layers did not really deteriorate machine utilizations.

In summary, with the analysis and discussion given above, one can conclude that the manufacturing system under study performs best when knowledge based control is employed at all levels of the control hierarchy. It is also clear that non-programmed system level control provides slightly better results than the system with no knowledge based control. On the other hand, the benefits of adding a knowledge based control at the shop control level can not be clearly observed on the main statistics of interest and traditional statistics. As noted previously, redesign of this knowledge base with more explicit and precise rules could possible give better results.

The other possible set of experiments with the model such as making replications with only shop knowledge base, with only work center knowledge base and other combinations to see the combined and/or stand alone effects of the knowledge bases is beyond the scope of this research effort and left as a future investigation area.

Another major advantage of the modeling and simulation framework developed in this research is the flexibility it brings into the whole simulation and modeling process. The operations of both physical and logical aspects of a manufacturing system modeled not only closely represents reality but is very easy to change. One main concept that is continuously brought up in the manufacturing planning and

control area but cannot be easily shown is the suboptimization of total system performance due to a collection of local optimals. This new simulation framework can easily be used as a demonstration tool to show that various local optimization rules and procedures of different control levels (or even different modules of the same level), when put together, deteriorates total system performance. It can also be a very useful tool to show how conflicting multiple objectives of different control levels or even within the same control level interact and affect global performance.

As noted earlier, the manufacturing system and the knowledge bases are artificial and used for demonstration purposes. But the main modules of the software and the framework with its modeling methodology are generic enough to be easily applied to most discrete part manufacturing systems.

Analytic Hierarchy Process

This section describes the application of the Analytic Hierarchy Process (AHP) to evaluate various aspects of the developed framework and the methodology against the conventional simulation paradigm. A detailed explanation of the AHP process will not be covered here and can be obtained from Saaty (1988). In simple terms, AHP is a multi-criteria decision methodology that utilizes structured pair wise comparisons among similar aspects of alternatives to reach a scale of preference. A more comprehensive application of AHP for simulation environment evaluation purposes can also be seen from the study done by Beaumariage (1990). This second reference compares object oriented simulation environments against traditional environments such as SLAM and SIMAN, and can also be referred to for a summarized guideline of the AHP application process. Thus, the process of developing an AHP model will not be covered here, but can be obtained from one of the sources mentioned above. Furthermore, since it is concluded that the object oriented environment is significantly better than traditional environments in Beaumariage's (1990) study, the direct advantages associated with the object oriented style is not included in the evaluation process of this research. Instead, a direct application of the process for the evaluation of the model and the modeling methodology will be outlined in the following sections.

The preliminary AHP model developed by the author was discussed, critiqued, and iterated by an AHP study group that consisted of Chuda Basnet, David Pratt, Philip Farrington (three Ph.D. candidates in industrial engineering at Oklahoma State University), and the author. Once the levels, the major aspects, and the criteria were finalized in terms of a set of nodes, the next task of the group was the definition of linkages between these nodes through an iterative process. After the links between nodes were agreed upon, the resulting preference matrices were formed and weighted by the group, again in an iterative manner. The following paragraphs gives a summary of the resulting levels, major aspects, criteria and weight matrices of the group's study.

Level 1 : Definition of the Problem

1.1 - Simulation paradigm : The problem on hand is the selection of the best simulation modeling methodology and the resulting model. The methodology in this context is interpreted as the whole process of conceptualizing and representing the system in terms of a simulation model along with underlying structures.

Level 2 : Main Aspects

2.1 - Model effectiveness : This is the model's capability of being used as a realistic decision support tool. That is to say, the aspects of the real system that can be represented in the model and the performance measures

that can be obtained closely express the real system. In addition, the model's ability to manage change, extension, reusability, and detail level are also considered as part of this aspect. This node links to node 1.1.

2.2 - Model developer's potency and modeling effort : This aspect of the decision problem addresses the capabilities that are associated with the model developer and the effort required to build a model. The model developer's activities heavily depends on the conceptualization of the model, and the tools and facilities provided by the modeling environment. The lower level criteria are evaluated either in terms of increasing the modeler's capability or decreasing the modeling effort required. This node also links to 1.1.

2.3 - Model execution performance : This is basically the computer time required to experiment with the model. This aspect is considered as one of the main factors due to rapid deterioration of model execution time performance as layers of knowledge based systems are added to the model. This node links to 1.1.

2.4 - Model's degree of correspondence to the real system : This aspect is very important for the model's acceptance as a valid tool for gaining insight about the real system. Depending on the desired level of detail in the system to be represented, this aspect evaluates how accurately the real system can be expressed in the model. Similar to the other nodes of this level, this node also

links to 1.1 to allow the relations defined at lower levels to factor into the final result.

Level 3 : Criteria Considered

3.1 - Formal modeling structures/modeling methodology : This criterion covers the underlying structures of the simulation paradigm and the modeling methodology dictated by those structures. In a sense, it is the science base of the simulation that gives the ability to answer questions like, how does one develop a model and why? This node links to model developer's potency and modeling effort, which is node 2.2.

3.2 - Model flexibility : This is the model's ability to express different aspects of the system as well as ease of model alteration and extension. The capability of developing models with different levels of detail without major model overhauls is also part of the flexibility criteria. This links to nodes 2.1, 2.2, and 2.4.

3.3 - Output provisions : This criterion represents the versatility of the data and the information from a simulation run. This includes the data collection facilities on physical and logical aspects of the system being modeled. This criterion has a strong influence on model effectiveness and therefore is linked to node 2.1.

3.4 - Execution speed : The cpu time required to run the simulation model represents the execution speed. This criterion interacts only with node 2.3, which is the model

execution performance aspect of a simulation.

3.5 - Physical, information, and control components : This is the simulation paradigm's ability to represent physical, information, and control components of the system under study in a modular fashion. This criterion increases the validity of the model, thereby promoting the credibility of the whole simulation study. This criterion links with all the aspects defined at level 2, except model execution performance. Therefore, it links to nodes 2.1, 2.2, and 2.4.

3.6 - Primitive modeling constructs : These are the basic building blocks of the model. The modularity and the variety of the constructs along with their expressiveness bring significant advantages to the whole simulation process. The primitive modeling constructs affect model developer's potency and model's degree of correspondence to the real system, and hence is linked to nodes 2.2 and 2.4.

3.7 - Non-programmed decision facilities : This criterion represents the simulation's ability to employ a hierarchical set of non-programmed decision support modules within the model. This is where mimicking the behavior of intelligent system entities that drive the entire operation of the real system comes into play. This criterion is considered to have an impact on all aspects defined at level 2, including model execution performance, and is linked to nodes 2.1, 2.2, 2.3, and 2.4.

Level 4 : Alternative Simulation Paradigms

4.1 - Conventional simulation paradigm : This is the traditional approach to simulation. The system is mainly conceptualized in terms of physical components with no explicit information or control modules attached. The logic that governs the model behavior is implicitly expressed through a set of generic (not system relevant) and abstract modeling constructs. Modeling in this approach is analogous to developing a computer program in a simulation language.

4.2 - Simulation framework proposed in this study : This approach includes the new modeling methodology and its underlying formalism. In this approach, the system (manufacturing system) is perceived in terms of a set of interacting physical, information, and control components. These highly uniform, modular and alterable components are the basic model building blocks. Simulation modeling in this approach is the process of tailoring these default constructs and defining the linkages among them to accurately represent a particular system's behavior.

Figure 24 shows the AHP hierarchical diagram. Tables II through XIII show the original weights of the AHP matrices agreed on by the study group.

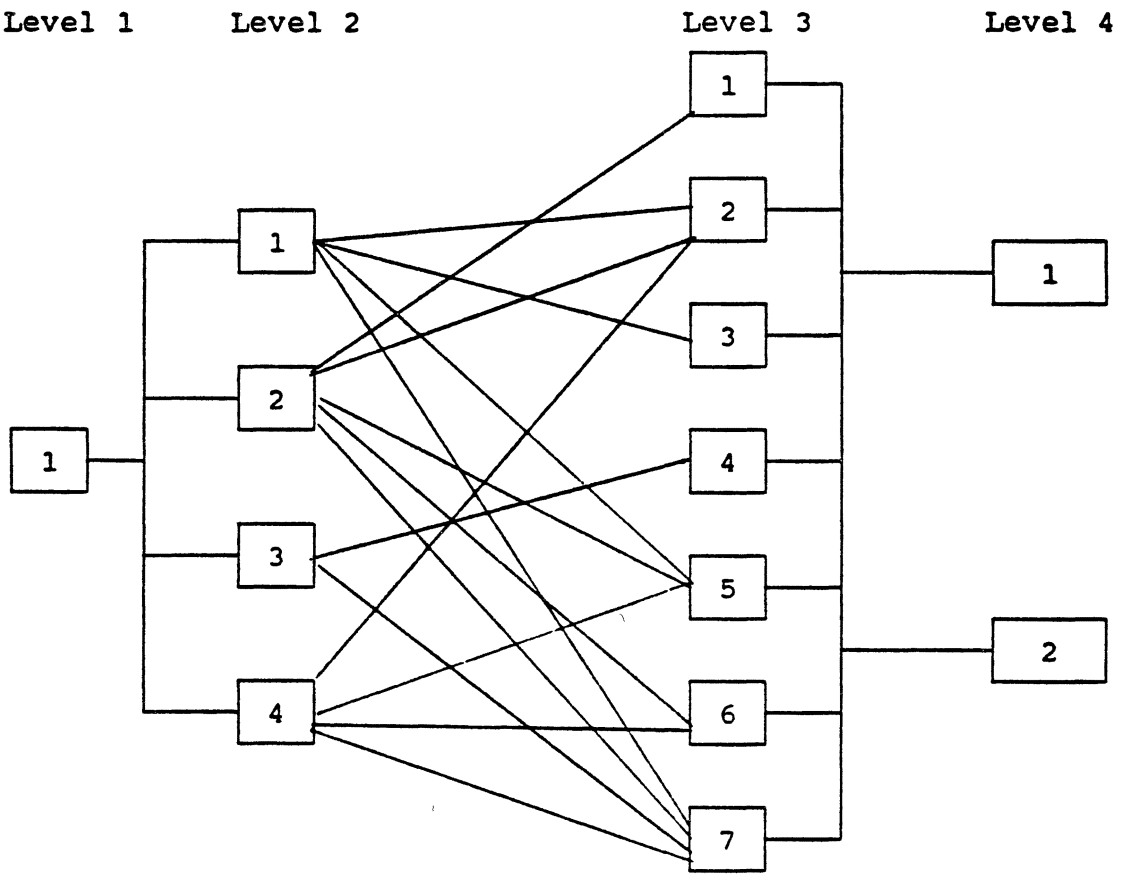


Figure 24, AHP Hierarchical Diagram

TABLE II

NODE 1.1 - BEST SIMULATION PARADIGM

Links from Lower Level:

1) Node 2.1 - Model effectiveness
2) Node 2.2 - Model developer's potency and modeling effort
3) Node 2.3 - Model execution performance
4) Node 2.4 - Model's degree of correspondence to the real system

Original weights

Col / Row	1	2	3	4
1	1.000	5.000	8.000	3.000
2	0.200	1.000	6.000	0.250
3	0.125	0.167	1.000	0.167
4	0.333	4.000	6.000	1.000

TABLE III

NODE 2.1 - MODEL EFFECTIVENESS

Links from Lower Level:

1) Node 3.2 - Model flexibility
2) Node 3.3 - Output provisions
3) Node 3.5 - Physical, information, and control components
4) Node 3.7 - Non-programmed decision facilities

Original weights

Col / Row	1	2	3	4
1	1.000	0.250	0.500	0.200
2	4.000	1.000	3.000	3.000
3	2.000	0.333	1.000	3.000
4	5.000	0.333	0.333	1.000

TABLE IV

NODE 2.2 - MODEL DEVELOPER'S POTENCY AND MODELING EFFORT

Links from Lower Level:

1) Node 3.1 - Formal model structures and modeling methodology
2) Node 3.2 - Model flexibility
3) Node 3.5 - Physical, information, and control components
4) Node 3.6 - Primitive modeling constructs
5) Node 3.7 - Non-programmed decision facilities

Original weights

Col / Row	1	2	3	4	5
1	1.000	0.333	0.333	0.333	3.000
2	3.000	1.000	4.000	2.000	3.000
3	3.000	0.250	1.000	0.333	4.000
4	3.000	0.500	3.000	1.000	4.000
5	0.333	0.333	0.250	0.250	1.000

TABLE V

NODE 2.3 - MODEL EXECUTION PERFORMANCE

Links from Lower Level:

1) Node 3.4 - Execution speed
2) Node 3.7 - Non-programmed decision facilities

Original weights

Col / Row	1	2
1	1.000	8.000
2	0.125	1.000

TABLE VI

NODE 2.4 - MODEL'S DEGREE OF CORRESPONDENCE TO REAL SYSTEM

Links from Lower Level:

1) Node 3.2 - Model flexibility
2) Node 3.5 - Physical, information, and control components
3) Node 3.6 - Primitive modeling constructs
4) Node 3.7 - Non-programmed decision facilities

Original weights

Col / Row	1	2	3	4
1	1.000	0.143	0.143	0.125
2	7.000	1.000	4.000	0.500
3	7.000	0.250	1.000	0.333
4	8.000	2.000	3.000	1.000

TABLE VII

NODE 3.1 - FORMAL MODELING STRUCTURES/ MODELING METHODOLOGY

Links to Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Col / Row	1	2
1	1.000	0.143
2	7.000	1.000

TABLE VIII

NODE 3.2 - MODEL FLEXIBILITY

Links from Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Col / Row	1	2
1	1.000	0.333
2	3.000	1.000

TABLE IX

NODE 3.3 - OUTPUT PROVISIONS

Links from Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Col / Row	1	2
1	1.000	0.333
2	3.000	1.000

TABLE X

NODE 3.4 - EXECUTION SPEED

Links from Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Row \ Col	1	2
1	1.000	5.000
2	0.200	1.000

TABLE XI

NODE 3.5 - PHYSICAL, INFORMATION, AND CONTROL COMPONENTS

Links from Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Row \ Col	1	2
1	1.000	0.111
2	9.000	1.000

TABLE XII

NODE 3.6 - PRIMITIVE MODELING CONSTRUCTS

Links from Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Col / Row	1	2
1	1.000	0.333
2	3.000	1.000

TABLE XIII

NODE 3.7 - NON-PROGRAMMED DECISION FACILITIES

Links from Lower Level:

1) Node 4.1 - Conventional modeling approach
2) Node 4.2 - New modeling paradigm

Original weights

Col / Row	1	2
1	1.000	0.143
2	7.000	1.000

The next step in the AHP procedure was the calculation of the relative weights of the decision elements. The spreadsheets developed by Beaumariage (1990) are used for this purpose. These spreadsheets calculated the priorities for each of the above matrices along with matrix consistencies (Appendix E). Then, after checking the

consistencies, these relative weights are aggregated through a series of matrix calculations to yield a solution to the problem. Table XIV shows the resulting final weights.

TABLE XIV

FINAL SOLUTION WEIGHTS

Conventional simulation paradigm	.203
New simulation paradigm	.797

The results of final weights obtained from AHP clearly indicate that the new simulation paradigm is preferable to the conventional paradigm in terms of the aspects and criteria considered in the AHP study. The conclusion reached in this AHP study is also consistent with Beaumariage's (1990) results, which were obtained using a different set of factors.

The example system modeled for validation of the formalism and the software developed shows the potential of the framework , although the manufacturing system modeled is a simple one and the software is a prototype. Even though not demonstrated, this new approach allows the simulation analyst to collect and analyze certain types of data such as

performance of different non-programmed control schemes and/or objects, and information processing capabilities of different objects and/or hierarchical levels. These are multi-criteria aspects of various decision problems that are not possible to study with traditional simulation paradigms. The formalism and the modeling methodology also dictate a uniform model. That is, different model developers who conceptualize the system in the same structured manner can come up with almost identical models. In contrast, the highly acclaimed and so called flexibility of the traditional approach results in several different models for the same real system giving highly different results.

The final weights obtained from the AHP study is due to significant advantages provided by the new simulation paradigm. The formal structures provide the basic constructs for modeling information, control, and physical aspects of the system independently and simultaneously, which is a considerable improvement over the traditional approach. The use of several knowledge bases interactively during the simulation and the uncertainty aspect allows more realistic and versatile representation of real systems.

CHAPTER X

CONCLUSIONS AND RECOMMENDATIONS

The goal of this research effort was the development of a cohesive framework that unifies presently segmented knowledge on discrete event simulation, object oriented modeling, and imbedded decision making processes of a system being modeled. This chapter summarizes the conclusions reached, research contributions, and recommendations of this study.

Conclusions

The specific problem area addressed in this study was the inadequacy of current simulation modeling approaches in providing a competent representation of information processing and decision making elements of a purposeful system. This problem was addressed within the context of a manufacturing system and five specific objectives were established for the investigation of the problem (also see chapter IV).

The first objective was to develop a formalism for discrete event simulation that integrates qualitative and quantitative aspects of a system in a structured form. To accomplish this objective, earlier studies that have been

done in this area were carefully studied and understood. Then, using the knowledge gained, a system was conceptualized as a set of sophistication levels which starts with the fact system and evolves into an intelligent system. A formalism that associates this concept with a typical manufacturing control hierarchy was defined. The developed formalism symbolically and explicitly expresses physical entities, data, information, knowledge, and intelligent control entities of a system. The structuring of the formalism in a consistent symbolic form was a major task and required several iterations. The developed formalism which explains the structure and behavior in a simulation model is accepted as adequate for the fulfillment of the first research objective. The satisfaction of this objective allowed the structuring of the software development at a later phase of the research.

In this study, the decision making processes are assumed to be the driving forces of a manufacturing system. Respectively, the second objective was the design of a prototype information/knowledge processing scheme that is consistent with the developed formalism. This objective was achieved through the definition of two different processing frameworks, identified as information processing and knowledge processing, each compatible with certain constructs of the formalism. Information processing is perceived as the manipulation of data and/or information and corresponds to programmed control in a manufacturing system.

Knowledge processing on the other hand, is perceived as the non-programmed decision making mechanism of a manufacturing system and is based on certain ideas borrowed from the AI field. The proposed framework effectively conceptualizes and expresses the embedded decision processes of a system in a simulation model. Specifically, the implementation successfully achieved the goal of "plug compatible decision modularity", in which the decision processes were modeled separately from the physical processes. There have been no similar achievement by other researchers reported in the literature.

The third objective was the development of a translation mechanism to communicate the goals and the tasks between different manufacturing control levels. Essentially, this objective is perceived as the preliminary work for the design of a generic manufacturing task language. The language-like scheme, based on certain formal language theory concepts, is developed to formalize the translation of orders communicated between hierarchical control levels. The manufacturing orders of various types are perceived as the goals for the receiving control level that need to be satisfied. The proposed translation mechanism, although a prototype scheme, conforms to the developed formalism, thereby resulting in the fulfillment of the third objective.

The fourth objective was the software implementation of the concepts developed for the fulfillment of objectives

one, two , and three in a simplified form. In order to achieve this objective, the Smalltalk-80 simulation environment was carefully studied and modified as required. The next step was the definition of the classes and message protocols to implement the formalism without making substantial modifications either in the formalism or in the scheduling and object coordination mechanism of Smalltalk-80. The classes defined in the hierarchy allows a more natural means of modeling for manufacturing systems and thus implies a new modeling methodology. In this scheme, the model developer conceptualizes a manufacturing system first in terms of physical components, then a set of modular control structures attached to those physical components. In addition, the model developer has the capability of easily redirecting material and information flow in the model. Therefore, this fourth objective is fulfilled by the definition and implementation of the class hierarchy presented in chapter VIII.

The fifth and final objective was the evaluation of the overall framework in terms of several aspects. Two different methods are identified for this purpose. First, the developed framework is demonstrated through an example to show the capabilities of the new approach. This was an informal and indirect evaluation approach and helped to observe practical aspects of the framework. It is observed that this new paradigm not only enriches the simulation environment but adds many desirable new characteristics to

the simulation in general. Although it is only a prototype, the developed software served well to realize several promising dimensions of the approach. The second method used for evaluation, which was direct and formal, was the application of Analytic Hierarchy Process to compare the developed framework to the conventional simulation paradigm. The criteria and the aspects considered in the AHP study as well as the weights defined for pair wise comparisons were designed with the help of an AHP study group. In its effort, the group considered possible future improvements in addition to the present states of both the paradigms. The result of the AHP clearly indicated that the developed formalism is an improvement over the conventional simulation approach. This conclusion of AHP was in accord with the experience gained through the example modeling study and successfully fulfilled the last objective.

Contributions

The main contribution of this research to the body of simulation knowledge is the development of the elementary foundation work that attempts to integrate fragmented yet closely related aspects involved in manufacturing system simulation. The undertaking of the formalism alone was a major task and oriented towards the development of a science base for simulation modeling. The nonexistence of a sound science base was identified as one of the major deficiencies of the present simulation approach at the beginning of the

research. This conventional approach focuses only on physical system aspects with some implicit representation of logical aspects. The proposed formalism on the other hand, provides the basic symbolic representations for simulation modeling and execution and outlines the elementary constructs for explicit representation of data, information, knowledge, and intelligence in an integrated form. The study on formalism alone, even in its present elementary form, alone is considered a significant contribution and will show its usefulness in conceptualizing complex manufacturing systems for simulation modeling purposes.

The fulfillment of objectives two and three also resulted in some preliminary original work on knowledge processing and language development within the context of manufacturing systems. The studies done in these two areas are introductory and demonstrate the use of well structured concepts of related fields like AI and Computer Science in simulation. The main contribution of this research in these areas is the establishment of the analogy between different study fields and incorporating certain new views into simulation modeling.

The defined Smalltalk-80 classes along with their hierarchy, even though requiring some refinements, lay down the basic structures and concepts for the software implementation. The defined generic classes conform to the formalism and the Smalltalk's scheduling and object coordination mechanism. In addition, the deficiencies of

the Smalltalk provided simulation classes were identified and several corrective actions were taken. The overall structure of the software and the uniformity of object definition and use in the proposed simulation methodology is considered significant progress. The example problem studied showed how detailed and powerful simulation can be when embedded decision making processes are explicitly expressed in the model.

The overall contribution of the research can be summarized as the identification and beginning of initial study in the areas of formalism, information/knowledge processing during simulation, and manufacturing task language along with a prototype software.

Recommendations

The areas investigated in this research were mainly virgin areas of research and require more examination and refinement for practical use in simulation. The manufacturing system simulation formalism is a promising further research area that will lay down the scientific base. Hopefully, further progress made in this field will promote a more structured modeling methodology than today's highly informal modeling practices. The definitions and basic symbolic constructs of the proposed formalism can be further polished for more elegant structural and behavioral model representation.

The outlined knowledge processing scheme is another area that requires advanced examination. Recent theoretical work in the AI field can be investigated and adopted to the problems addressed regarding the activities of intelligent manufacturing system entities. Even an elementary approach, similar to the one utilized in the software implementation of this research, can vastly increase the capabilities of models and the insight gained from the simulation.

It is clear that the manufacturing task language will be an essential part of the framework for appropriate communication of intelligent entities. The translation of system goals and their interpretation will all depend on the effectiveness of the formal manufacturing language used. Further study in this area will also help to formalize the casual language used in present manufacturing systems and its adaptation to simulation modeling.

The immediate benefits of the developed framework will most likely come from enhancements done on the prototype software. The graphical user interface classes attached to system and product structure definition classes will significantly improve the value of the software. The second set of expansions will target the tailoring of default programmed and/or non-programmed control, material flow control, and communication classes with a user friendly interface. The Humble knowledge base rule definition method can also be identified as a possible area for a friendlier interface.

BIBLIOGRAPHY

Abrams, M. "The Object Library for Parallel Simulation (OLPS)." Proceedings of the 1988 Winter Simulation Conference, San Diego, CA, pp. 210-219.

Adelsberger, H. H., U. W. Pooch, R. E. Shannon, and G. N. Williams. "Rule Based Object Oriented Simulation Systems." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986, pp. 107-112.

Adelsberger, H. H. and G. Neumann. "Goal Oriented Simulation Modeling Using Prolog." Proceedings of the 1985 SCS Conference on Modeling and Simulation on Microcomputers, San Diego, CA.

Adiga, S. "Software Modelling of Manufacturing Systems: A Case for an Object-Oriented Programming Approach." Working Paper, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA, 1986.

Banks, J. and J. S. Carson. Discrete-Event System Simulation. Prentice-Hall, Englewood Cliffs, NJ, 1984.

Baskaran, V. and R. Reddy. "An Introspective Environment for Knowledge Based Simulation." Proceedings of the 1984 Winter Simulation Conference, Dallas, TX, pp. 645-651.

Beaumariage, T. G. "Investigation of an Object Oriented Modeling Environment for the Generation of Simulation Models." Unpublished Ph.D. Dissertation, School of IE and Mgmt., Oklahoma State University, Stillwater OK, 1990.

Begg, I. M. and R. C. Worsley. "AUDITION - An Intelligent Simulation Environment." Simulation and AI, The Society for Computer Simulation International, San Diego, CA, 1989.

Bezivin, J. "Timelock: A Concurrent Simulation Technique and its Description in Smalltalk-80." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 503-506.

Braizer, M. K. and R. E. Shannon, "Automatic Programming of AGVS Simulation Models." Proceedings of 1987 Winter Simulation Conference, Atlanta, GA.

Bunn, D. W. Applied Decision Analysis. McGraw-Hill, San Fransisco, CA, 1986.

Burns, J. R. and J. D. Morgeson. "A Methodology for Formulation of Knowledge Based Simulation Models." Information and Decision Technologies, 1988, Vol. 14, pp. 15-30.

Burns, J. R. and J. D. Morgeson. "An Object-Oriented World-View for Intelligent, Discrete Next-Event Simulation." Management Science, December 1988, Vol. 34, pp. 1425-1440.

Burns, J. R., J. D. Morgeson and H. W. Egdorf. "A Definition of Software Requirements and Design Specifications for Fabrication of Intelligent Discrete-Event Simulation Models." Los Alamos National Laboratory, Los Alamos, New Mexico, 1986.

Castillo, D., M. McRoberts, S. Green and B. Sieck. "HSKB : An Architecture for Embedded Reasoning in Simulation Systems." Simulation and AI, The Society for Computer Simulation International, 1989, pp. 41-46.

Cellier, F. E. and B. P. Zeigler. "AI's Role in Control of Systems: Structural and Behavioral Knowledge." Simulation in CIM and AI Techniques, The Society for Computer Simulation, San Diego, CA, 1987, pp. 165-171.

Charniak, E. and D. McDermott Artificial Intelligence. Addison-Wesley: Reading, MA, 1987.

Chryssolouris, G. and I. Gruenig. "On a Database Design for Intelligent Manufacturing Systems." International Journal of Computer Integrated Manufacturing. 1989, Vol.1, No. 13, pp. 171-184

Cleary, J. and A. Dewar "Interpreters for Logic Programming." Proceedings of SCS Conference on Simulation in Strongly Typed Languages, San Diego, CA, 1984.

Clocksin, W. F. and C. S. Mellish. Programming in Prolog. Springer-Verlag, Berlin, W. Germany, 1984.

Cox, B. Object Oriented Programming: An Evolutionary Approach. Addison-Wesley, Reading, MA, 1986.

Cox, B. and B. Hunt. "Objects, Icons, and Software-IC's." Byte, August, 1986, pp. 161-176.

Digitalk, Inc. Smalltalk/V Tutorial and Programming Handbook. Digitalk, Inc., Los Angeles, CA, 1986.

Dahl, O. J. and K. Nygaar. "SIMULA-An Algol Based Simulation Language." Communications of the ACM, 1966, Vol. 9.

Doman, A. "Object-Prolog: Dynamic Object-oriented Representation of Knowledge." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation, San Diego, CA, 1988.

Endesfelder, T. and H. Tempelmeier. "The SIMAN Module Processor - A Flexible Software Tool for the Generation of SIMAN Simulation Models." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation, San Diego, CA, 1987, pp. 38-43.

Eversheim, W. "Graphic Interactive Simulation for the Planning of Manufacturing Systems." Journal of Manufacturing Systems, Vol. 6, No. 2, pp. 151-156.

Farnsworth, K. D., V. B. Norman, and T. A. Norman. "Integrated Software for Manufacturing Simulation." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 195-201.

Fishwick, P. A. "A Study of Terminology and Issues in Qualitative Simulation." Simulation, January, 1989a, pp. 5-9.

Fishwick, P. A. "Qualitative Methodology in Simulation Model Engineering." Simulation, March, 1989b, pp. 95-101.

Fishwick, P. A. "Qualitative Simulation: Fundamental Concepts and Issues." AI and Simulation: The Diversity of Applications. The Society for Computer Simulation International, San Diego, CA, 1988.

Ford, D. R., B. J. Schroer, and R. Daughtrey. "An Intelligent Modeling System for Simulation Manufacturing Processes." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 525-529.

Futo, I., I. Papp and J. Szeredi. "The Microcomputer Version of TC-Prolog." Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986.

Genesereth, M. R. and N. J. Nilsson. Logical Foundations of Artificial Intelligence. Morgan-Kaufman, Los Angles, CA, 1987.

Goldberg, A. and D. Robson. SMALLTALK-80: The Interactive Programming Environment. Addison-Wesley, Reading, MA, 1989.

Goldberg, A. and D. Robson. SMALLTALK-80: The Language and Its Implementation. Addison-Wesley, Reading, MA, 1989.

Gordon, R. F., E. A. MacNair, K. J. Gordon, and J. F. Kurose. "A Visual Programming Approach to Manufacturing Modeling." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 465-471.

Griesmeyer, J. M. "Generalized Simulation Environment for Factory Systems." Tools for the Simulation Profession 1989, The Society for Computer Simulation, San Diego, CA, 1988, pp. 20-29.

Haddock, J. "A Simulation Generator for Flexible Manufacturing Systems Design and Control." IIE Transactions, Vol. 20, No. 1, pp. 22-31.

Helman, D. H. and A. Bahuguna. "Explanation Systems for Computer Simulations." Proceedings of the 1986 Winter Simulation Conference, Washington D.C., WA, pp. 453-459.

Henriksen, J. O. "The Integrated Simulation Environment (Simulation Software of the 1990's)." Operations Research, Vol. 31, No. 6, 1983, pp. 1053-1072.

Higdon, J. "Planning a New Material Handling System." Industrial Engineering, November, 1988, pp. 55-59.

Hilton, M. L. "A Multi-level Event Scheduling Mechanism for Supporting Intelligent Objects." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation: San Diego, CA, 1988, pp. 127-130.

Hong, S. C., J. K. Cochran. and G. T. Mackulak. "Specification of An Architecture for Intelligent Simulation Environments." Simulation and AI. The Society for Computer Simulation International, San Diego, CA, 1989.

Hsu, C. and C. Skevington. "Integration of Data and Knowledge in Manufacturing Enterprises: A Conceptual Framework." Journal of Manufacturing Systems, Vol. 6, No. 4, pp. 277-285.

Kaehler, T. and D. Patterson. "A Small Taste of Smalltalk." Byte, August, 1986, pp. 145-159.

Kitzmiller, C. T. "Simulation and AI: Coupling Symbolic and Numeric Computing." AI and Simulation: The Diversity of Applications. The Society for Computer Simulation International, San Diego, CA, 1988.

Khoshnevis, B. and A. Chen. "An Automated Simulation Modeling System based on AI Techniques." Simulation and AI, Proceedings of the Conference on AI and Simulation, The Society for Computer Simulation, San Diego, CA, 1987, pp. 87-91.

Khoshnevis, B. and A. Chen. "An Expert Simulation Model Builder." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986, pp. 129-132.

Khoshnevis, B. and W. M. Austin. "An Intelligent Interface for System Dynamics Modeling." Simulation and AI, Proceedings of the Conference on AI and Simulation, The Society for Computer Simulation: San Diego, CA, 1987, pp. 81-86.

Kimbler, D. L. and B. A. Watford. "Simulation Program Generators: A Functional Perspective." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation, San Diego, CA, 1988, pp. 133-136.

King, C. U., and E. L. Fisher. "Object-Oriented Shop-Floor Design, Simulation and Evaluation." 1986 Fall Industrial Engineering Conference Proceedings, Institute of Industrial Engineers, Boston, MA, 1986, pp. 131-137.

King, C. U., S. S. Adams, and E. L. Fisher. "Representation of Manufacturing Entities." Intelligent Manufacturing: Proceedings from the First International Conference on Expert Systems and the Leading Edge in Production Planning and Control, Charleston, SC, 1987, pp. 77-91.

Klahr, P., W. S. Faught and G. R. Martin. "Rule Oriented Simulation." Proceedings of the 1980 International Conference on Cybernetics and Society, IEEE, Cambridge, MA, pp. 350-354.

Klir, G. J. "General Systems Framework for Inductive Modeling." Simulation and Model Based Methodologies: An Integrative View. NATO ASI Series, Springer-Verlag, Berlin, West Germany, 1984.

Knapp, V. E. "The Smalltalk Simulation Environment, Part II." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 146-151.

Kreutzer, W. "A Modellers's Workbench - Simulation Based on the Desktop Methaphor." Artificial Intelligence and Simulation: The Diversity of Applications, The Society for Computer Simulation, San Diego, CA, 1988, pp.43-48.

Kreutzer, W. System Simulation: Programming Styles and Languages. Addison-Wesley, Reading, MA, 1986.

Kumar, A. and S. Y. W. Su. "Object Manipulation in an Object-Oriented Semantic Association Model (OSAM)." Manufacturing International '88, Atlanta, GA, 1988.

Law, A. M. and W. D. Kelton. Simulation Modeling and Analysis. McGraw-Hill, New York, NY, 1982.

Mellichamp, J. M. and A. F. Wahab "An Expert System for FMS Design." Simulation, 1987, Vol. 48, No. 5, pp. 201-208.

Miller, J. A., O. R. Weyrich, Jr., and D. Suen. "A Software Engineering Oriented Comparison of Simulation Languages." Tools for the Simulation Profession 1989, The Society for Computer Simulation, San Diego, CA, 1988, pp. 97-104.

Murray, K. J. and S. V. Sheppard. "Automatic Model Synthesis: Using Automatic Programming and Expert Systems Techniques toward Simulation Modeling." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 534-543.

Nadoli, G. and Biegel J. "Intelligent Agents in the Simulation of Manufacturing Systems." Advances in AI and Simulation, The Society for Computer Simulation, San Diego, CA, 1989.

Nyen, P. A. "A Comprehensive Environment to Object Oriented Simulation of Manufacturing Systems." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation, San Diego, CA, 1987, pp. 21-25.

Odubiyi, J. B. "A Framework for Establishing the Operational Capability for Expert Systems with Model or Scenario Based Reasoning." AI and Simulation: The Diversity of Applications, The Society for Computer Simulation International, San Diego, CA, 1988.

O'Keefe, R. M. "What is Visual Interactive Simulation? (and is there a Methodology for doing it right?)." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 461- 464.

O'Keefe, R. M. "Simulation and Expert Systems: A Taxonomy and Some Examples." Simulation, 1986, Vol. 46, No. 1, pp. 10-16.

Oren, T. I. "AI and Simulation: From Cognitive Simulation Toward Cognizant Simulation." Simulation, Vol. 48, No. 4, April 1987, pp. 129-130.

Oren, T. I. "Knowledge Bases for An Advanced Simulation Environment." Intelligent Simulation Environments. The Society for Computer Simulation, San Diego, CA, 1986.

Oren, T. I., B. P. Zeigler, and M. S. Elzas, editors. Simulation and Model-Based Methodologies: An Integrative View. NATO ASI Series, Springer-Verlag, Berlin, West Germany, 1984.

Oren, T. I. "GEST - A Modeling and Simulation Language Based on Systems Theoretic Concepts." Simulation and Model-Based Methodologies: An Integrative View. NATO ASI Series, Springer-Verlag, Berlin, West Germany, 1984.

Oren, T., and K. Aytac. "Architecture of MAGEST: A Knowledge-based Modeling and Simulation System." Simulation in Research and Development, Elsevier Science Publishers, North-Holland, 1985, pp. 99-109.

Pascoe, G. A. "Elements of Object-Oriented Programming." Byte, August 1986, pp. 139-144.

Pazirandeh, M. and J. Becker. "Object Oriented Performance Models with Knowledge-Based Diagnostics." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 518-524.

Pichler, F. "Symbolic Manipulation of System Models." Simulation and Model-Based Methodologies: An Integrative View. NATO ASI Series, Springer-Verlag, Berlin, West Germany, 1984.

Portier, F. J. "Implementing the Product Automaton Formalism." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 544-553.

Pritsker, A. A. B. Introduction to Simulation and SLAM II, third edition. John Wiley & Sons, New York, 1986.

Radiya, A. and R. G. Sargent. "Logic Programming and Discrete Event Simulation." Simulation and AI, The Society for Computer Simulation, San Diego, CA, 1987.

Radiya, A. and R. G. Sargent. "A New Formalism for Discrete Event Simulation." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 554-558.

Reddy, R. "Epistomology of Knowledge Based Simulation." Simulation, 1988, Vol. 48, No. 4, pp. 162-166.

Reddy, R., M. S. Fox, N. Husain and M. McRoberts. "The Knowledge Based Simulation System." IEEE Software, March 1986, pp. 26-37.

Reilly, K., W. T. Jones, and P. Dey. "The Simulation Environment Concept: Artificial Intelligence Perspectives." Artificial Intelligence and Simulation, The Society for Computer Simulation, San Diego, CA, 1985, pp. 29-34.

Roberts, D. S. "A Perspective on Object-Oriented Simulation." Proceedings of the 1988 Winter Simulation Conference, San Diego, CA, pp. 282-286.

Robertson, P. "A Rule Based Expert Simulation Environment." Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986.

Rodrigues, A. J. "Structure and Logic in Knowledge Based Representations of Systems Simulation Models." European Journal of Operations research, 1988, Vol. 37, pp. 120-126.

Rozenblit, J. W. "Systems Theory Instrumented Simulation Modeling." Proceedings of the 1988 Winter Simulation Conference, San Diego, CA, pp. 282-286.

Rozenblit, J. W., T. G. Kim and B. P. Zeigler. "Towards an Implementation of a Knowledge Based System Design and Simulation Environment." Proceedings of the 1988 Winter Simulation Conference, San Diego, CA, pp. 226-230.

Rozenblit, J. W. and B. P. Zeigler. "Concepts for Knowledge Based System Design Environments." Proceedings of the 1985 Winter Simulation Conference, San Fransisco, CA, pp. 223-231.

Ruiz-Mier, S. and J. Talavage. "A Hybrid Paradigm for Modeling of Complex Systems." Simulation, 1987, Vol. 48, No. 4, PP. 135-141.

Ruiz-Mier, S. and J. Talavage. "Towards a Knowledge Based Network Simulation Environment." Proceedings of the 1985 Winter Simulation Conference, San Fransisco, CA, pp. 232-236.

Saaty, T. L. Decision Making: The Analytic Hierarchy Process. RWS Publications, Pittsburgh, PA, 1988.

Sathi, N., M. Fox, V. Baskaran, and J. Bouer. "An Artificial Intelligence Approach to the Simulation Life Cycle." A Technical Brief, Carnegie Group, Inc., Pittsburgh, PA, 1987.

Schriber, T. J. "The Nature and Role of Simulation in the Design of Manufacturing Systems." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation: San Diego, CA, 1987, pp. 5-18.

Schroer, B. J. and F. T. Tseng. "Modeling Complex Manufacturing Systems using Simulation." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 677-682.

Seliger, G., B. Viehweger, B. Weineke-Toutaoui and S. R. Kommana "Knowledge Based Simulation of FMS." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation, San Diego, CA, 1987.

Shannon, R. E. "Knowledge Based Simulation Techniques for Manufacturing." International Journal of Production Research, 1988, Vol. 26, No. 5, pp. 953-973.

Shannon, R. E. "Models and Artificial Intelligence." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 16-24.

Shannon, R. E. Systems Simulation: The Art and Science. Prentice-Hall: Englewood Cliffs, NJ, 1975.

Shaw, M. L. and B. R. Gaines. "A Framework for Knowledge Based Systems Unifying Expert Systems and Simulation." Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986.

Snyder, J. and G. T. Mackulak. "Intelligent Simulation Environments: Identification of the Basics." Proceedings of the 1988 Winter Simulation Conference, San Diego, CA, pp. 357-363.

Spiegel, J. R. and D. B. LaValle. "Using an Expert System to Drive a Simulation Experiment." AI Papers, The Society for Computer Simulation, San Diego, CA, 1988.

Szpakowicz, S., Kersten G. and Koperczak Z. "Simulating Decision Processes in the Rule-based Paradigm." Advances in AI and Simulation, The Society for Computer Simulation, San Diego, CA, 1989.

Thomas, D. "The Time/Space Requirements of Object-Oriented Programs." Journal of Object Oriented Programming, March/April, 1989, pp. 71-73.

Thomasma, T. and O. M. Ulgen. "Hierarchical, Modular Simulation Modeling in Icon Based Simulation Program Generators for Manufacturing." Proceedings of the 1988 Winter Simulation Conference, San Diego, CA.

Thomasma, T. and O. M. Ulgen. "Modeling of a Manufacturing Cell using a Graphical Simulation System Based on Smalltalk-80." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 683-691.

Ulgen, O. M. and T. Thomasma. "A Graphical Simulation System in Smalltalk-80." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation: San Diego, CA, 1987, pp. 53-58.

Unger, B., A. Dewar, J. Cleary, and G. Birtwistle. "A Distributed Software Prototyping and Simulation Environment: JADE." Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986, pp. 63-71.

Van der Meulen, P. "Development of an Interactive Simulator in Smalltalk." Journal of Object Oriented Programming, January/February 1989, Vol. 1, No. 5, pp. 28-51.

Vaucher, J. G. and G. Lapalme. "Process Oriented Simulation in Prolog." Simulation and AI, The Society for Computer Simulation, San Diego, CA, 1987.

Vesterager, J., K. E. Wichmann, R. E. Young, and J. Heide. "Simulation Uses in CIM Development." Simulation in CIM and Artificial Intelligence Techniques, The Society for Computer Simulation, San Diego, CA, 1987, pp. 95-100.

Vollmann, T. E., W. L. Berry, D. C. Whybark. Manufacturing Planning and Control Systems. Richard D. Irwing, Inc., Homewoods, Illinois, 1984.

Wales, F. J. and P. A. Luker. "An Environment for Discrete Event Simulation." Intelligent Simulation Environments, The Society for Computer Simulation, San Diego, CA, 1986, pp. 58-62.

Wallace, J. C. "The Control and Transformation Metric: Toward the Measurement of Simulation Model Complexity." Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987, pp. 597-603.

Wyvill, B. "Current Trends in Graphics and Animation." Artificial Intelligence, Graphics, and Simulation, The Society for Computer Simulation, San Diego, CA, 1985, pp. 49-53.

Xerox Special Information Systems. Humble Reference Manual, 1987, Xerox Corporation.

Zeigler, B. P. "Hierarchical, Modular Discrete-Event Modeling in an Object-Oriented Environment." Simulation, 1987, Vol. 49, No. 5, pp. 219-230.

Zeigler, B. P. "System Theoretic Representation of Simulation Models." IIE Transactions, March 1984.

Zeigler, B. P. Multifacetted Modelling and Discrete Event Simulation. Academic Press, London, 1984.

Zeigler, B. P. "Structures for Model Based Systems." Simulation and Model-Based Methodologies: An Integrative View. NATO ASI Series, Springer-Verlag, Berlin, West Germany, 1984.

Zeigler, B. P. Theory of Modelling and Simulation. Robert E. Krieger Publishing Co., Malabar, FL, 1976.

VITA

Seref Cem Karacal

Candidate for the Degree of

Doctor of Philosophy

Thesis: THE DEVELOPMENT OF AN INTEGRATIVE STRUCTURE FOR DISCRETE EVENT SIMULATION, OBJECT ORIENTED MODELING AND EMBEDDED DECISION PROCESSES

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Erzurum, Turkey, July 8, 1958, the son of Orhan and Gonul Karacal. Married to Zeynep Izgu on January, 24, 1984.

Education: Graduated from Deneme Lisesi, Ankara, Turkey, in June, 1976; received Bachelor of Science degree in Industrial Engineering form Middle East Technical University, Ankara, Turkey, 1982; received Master of Science degree from Oklahoma State University, 1986; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in May, 1991.

Professional Experience: Teaching Assistant, Industrial Engineering Department, Middle East Technical University, August 1982 to December, 1982; Manufacturing Engineer, Guris Machinery, Ankara, Turkey, January, 1983, to December, 1983; Graduate Assistant, Oklahoma State University, Industrial Engineering Department, August, 1986, to February, 1990; Assistant Professor, Rochester Institute of Technology, March, 1990, to present.